1978

Evaluation of Access Paths in a Relational Database System

S. Bing Yao

David De Jong

Report Number:
78-280

EVALUATION OF ACCESS PATHS IN A RELATIONAL DATABASE SYSTEM

S. Bing Yao
David De Jong

August, 1978
TR 280

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

De Jong,David Peter.MS.,Purdue University,May1978.Evaluation of Access Paths in a Relational Database System.Major Professor: S. B. Yao.

The motivation for and concepts of a relational database organization are briefly reviewed, and a general model of access paths for a relational database is described. As a tool for access path design and selection a series of equations are developed for the model to analyze the relative costs of the various paths. Comparisons show consistancy of these path costs with previous results and illuminate areas of improvement over earlier models. Sample tests illustrate the model's usefulness in comparing access path performances under various circumstances.

I. INTRODUCTION

The performance of a database system in accessing its data depends largely upon how well the organization of the data is suited to the types of access that are requested. No single database organization is "best" for retrieving data under all usage circumstances. For example, a sequential file organization works very well for sequential processing, but tree or hashing structures are better for randomly accessed data. In addition, logical relationships, or "links" (eg. hierarchies), may be defined among the data items. Since data is likely to be retrieved in accordance with these relationships, a physical organization patterned after the logical relationships (eg. hierarchical sequence) is desirable. So three factors which affect database performance are the type of access, or query, that the user requests, the logical relationships defined on the data, and the physical organization of the data.

An access path into a database is a series of steps which must be taken in order to search the database and retrieve the data requested by the user. These steps may include such operations as searching indexes, tracing linked lists, or sequentially scanning the data for the requested information. An obvious goal of a database design is to

provide access paths which retrieve the data efficiently. So the three performance factors mentioned above must be taken into consideration in designing access paths into the database. Since each of these factors offers many alternatives for consideration, the access path designer is faced with a myriad of choices. He cannot expect to make optimal choices without some analytic tools to help him evaluate, in advance, the performance of the alternatives before him.

For this reason several formal models of database organization have been proposed. The first models dealt with only physical organizations, while later models included query parameters. Only recently have logical relationship parameters been added. Hsiao and Harary [7] developed a model for single level, indexed database organizations, which was able to represent list structures of all lengths. A model suggested by Severance [11] added a second parameter to allow for sequential file organization. It provided representation for the various combinations of sequential and list structures. Cardenas [4] presented a very detailed model of the inverted file organization. Its parameters were also able to capture some of the query characteristics. A generalized, n-parameter model for access cost analysis was introduced by Yao [13]. The previously mentioned physical organizations were shown to be special cases of this model. Parameters for more complex

index structures as well as for query characteristics were included. A major limitation, however, in all of these models was that their applicability was constrained to single level (ie. flat file) database organizations. Logical relationships, or "links", among the data, which occur in many current database applications, were not able to be modeled.

This drawback has been indirectly addressed by Blasgen [3], Rothnei [9], Astrahan [1,2], Smith [12], and Precherer [8] in their works on implementing the relational model of database organization, a concept which we describe in detail later in this paper. Relations essentially represent sets of data, and so logical relationships, such as hierarchies and networks, may be defined among these sets. Blasgen's work was designed to evaluate and compare the "costs" of a selected group of access methods for the relational database. Parameters were included for each of the three major factors mentioned earlier: query characteristics, logical relationships, and physical organization. The obvious limitation, though, is that the results apply only to those certain access methods which were chosen, and which are by no means exhaustive.

The major goal of this work, then, has been the development of a more general cost model for access methods in a relational database, which calculates the "cost" of retrieving data when given parameters describing the desired

access method, query characteristics, logical relationships, and physical organization of the database. The ground work for such an access model has been laid by Yao [15]. In building on this foundation we have concentrated both on cost analysis of access paths, and on the resulting implementation and testing of the completed model. We have chosen the unit of cost to be the number of secondary storage accesses required to retrieve the data requested by a query.

In the remainder of this paper we first describe the relational database framework in greater detail. Then we present the access path model and its associated cost equations. Finally, we demonstrate the model's usefulness with examples and with comparisons with several previous results.

II. DATABASE FRAMEWORK: THE RELATIONAL MODEL.

A. ADVANTAGES.

In the early 1970's Codd introduced the relational model as an alternative framework for database organization. Although the concept remains largely in the experimental stage, many of the advantages which Codd suggested are already being proven. A major advantage which the relational system provides over previous systems is a much greater degree of independence of the user from the system. Codd mentioned three ways in which this is so. First, the user is independent of the ordering of the data. His view of the data, and thus his application program, are unaffected by modifications of the physical ordering of the data. The user is also independent of the various index structures built by the system for data retrieval. If the indexes are modified during the life of the database, the user is not affected. Thirdly, access paths to the data are transparent to the user. He tells the system which data to find, but does not need to specify how to find it. We see that the user is quite far removed from the physical organization of the data. This interface is now handled by part of the database system itself, often called the database management system (DBMS).

A second advantage of the relational model is its capacity to provide a "user friendly" interface between the user and the DBMS. The theory of relations has a strong mathematical background. This foundation has proven that a very small and concise set of operations on relations is all that is needed to obtain any subset of the data in the database. Further, this set of operations (in the form of either a relational algebra or relational calculus) is very logical in nature, appealing to both programming and non-programming users. Any user programming language built around this set of operations (called a relational data manipulation language) provides a high level user interface because of its independence from the physical organization of the database. Unfortunately, most of the attempts so far to implement such a language have resulted in rather "unfriendly" interfaces. We trust, though, that with more attention being given to this problem, improvements are not far away.

A third advantage of the relational model is its ability to implicitly deal with logical relationships among the data. Data items which are logically related, such as by having a common value, can be paired up even though they are not either physically close or connected by physical links. This point is discussed more thoroughly below.

EMPLOYEE

E#	ENAME	SAL	DEPT#
101	Smith	10000	1
102	Drasin	17000	1
103	Farley	13000	2
104	Veen	10000	3

PROJECT

PROJ-ID	DEPT#	MONTH-DUE
P-1	1	May
P-2	3	April
P-3	3	May
P-4	1	June

Figure 1. Two Relational Files.

B. DEFINITIONS.

We now wish to define in more detail the relational terminology from the viewpoint of a database system (as opposed to its mathematical background). Each logical record in a database is used to describe an entity, such as an employee. The characteristics of the entity which the record identifies, such as the employee's name and salary, are called its attributes. The actual data, such as "R. Smith" and "15,000" are values for the attributes. If m attributes are described, then the record consists of the m values for those attributes. A file is then defined to be a set of records which all describe the same set of attributes. The number of records in a file determines the file's size. Figure 1 shows two files called "EMPLOYEE" and "PROJECT" with attributes E#, ENAME , SAL and DEPT#; and PROJ-ID, DEPT#, and MONTH-DUE, respectively.

C. RELATIONAL OPERATIONS.

Any subset of a relational database can be obtained by repeated applications of these following three operations.

RESTRICTION. The specification of a simple restriction is of the form (attribute op value), where op is taken from [<,>,=,≥,≤]. It effectively searches the file for those records having the given value for the attribute. If, for example, we place the restriction (DEPT# = 1) on the

EMPLOYEE file in Figure 1, we obtain the first two records from the file. Compound restrictions may use the logical operators ^ (intersection) and ˇ (union) in order to combine simple restrictions. As an example, the restriction (DEPT# = 1) ^ (SAL < 15000) would pick only the first record out of the EMPLOYEE file.

PROJECTION. A projection essentially chooses a set of columns from a file. The columns to be picked out are specified by a list $(A_1, A_2, \ldots A_k)$ of attributes of the file. The projection (E#,ENAME) on the EMPLOYEE file creates the file shown in Figure 2.

E#	ENAME
101	Smith
102	Drasin
103	Farley
104	Veen

Figure 2. The projection (E#,ENAME) on the EMPLOYEE file.

When columns are chosen from a file in this way, it is possible to end up with duplicate records in the resulting

file. This would happen with the EMPLOYEE file if we chose to project only the DEPT# column. For this reason, the projection operation also looks for and eliminates any duplicate records. Figure 3 shows the result of the projection (DEPT#).

DEPT#
1
2
3

Figure 3. The projection (DEPT#) on the EMPLOYEE file.

EQUI-JOIN. The join operation is performed on two files which have an attribute, A, in common. Referring again to Figure 1, note that both files have an attribute for department number. If a value, v, of attribute A occurs in both files, then each record with value v is said to participate in the join. Join value "1" is an example, indicating that the first two records of EMPLOYEE and the first and third records of PROJECT are among the

participating records. Each such record with value v in the first file is concatenated with each such record having value v in the second file. Thus the first and third records of PROJECT are joined with each of the first two records of EMPLOYEE. "3" is another department number appearing in both files, so the fourth record of EMPLOYEE and the second and third records of PROJECT also participate in the join. We have then, in figure 4, the result of a join of the two files over the attribute DEPT#.

E#	ENAME	SAL	DEPT#	PROJ-ID	MONTH-DUE
101	Smith	10000	1	p-1	May
101	Smith	10000	1	p-4	June
102	Drasin	17000	1	p-1	May
102	Drasin	17000	1	p-4	June
104	Veen	10000	3	p-2	April
104	Veen	10000	3	p-3	May

Figure 4. The join of EMPLOYEE and PROJECT files over DEPT#.

Notice that for any join value the join operation does not restrict the number of records which may participate from

either file. Two records from each file were joined on the value DEPT# = 1. This shows that the relational database model has a powerful way of dealing with such "many-to-many" logical relationships in the data.

D. QUERY LANGUAGES.

Query languages implementing the relational operators fall into two categories: those based on relational calculus and those on relational algebra. Queries in an algebraic language are procedural. They specify how to search the database by indicating which relational operators are to be applied to which files. Relational calculus, on the other hand, is non-procedural. Through the use of quantifiers such as "for all"(∀) and "there exists"(∃), it specifies what subset is to be retrieved from the database. The DBMS must infer from such a query which operations should be used in order to find that subset. Numerous relational query languages have been proposed. Those based on relational calculus include DSL ALPHA and MORIS. Algebraic languages include MacAIMS, IS/1, RDMS, QUEL, and SEQUEL. For a brief comparison of these languages, the reader is referred to chapter 8 of Date [6].

E. A GENERAL QUERY.

For the purposes of the access path model described in this paper, we consider a generalized, two variable query, which utilizes each of the relational operations. The query deals with two files. It places a restriction on each file, joins the files over a common attribute, and projects a subset of the columns. As an example we refer again to Figure 1 with the following query:

> Find the employee name and project-id (projections) of all employees whose salary is 10000 (restriction on EMPLOYEE), and who work in departments having projects (join) due in May (restriction on PROJECT).

The result of this query is is shown if Figure 5.

ENAME	PROJ-ID
Smith	P-1
Drasin	P-1
Veen	P-3

Figure 5. The result of a sample query.

F. STORAGE ORGANIZATION.

We have stated that the physical organization of the database is an important factor in the cost of data retrieval. Three organization methods which can be implemented in the relational database system, as well as in hierarchical and network systems, have been included as parameters in our access cost model. They are indexing, linking, and clustering.

INDEXING. An index, such as one might find in the back of a textbook, is used to point to all the pages in the book where a particular word may be found. Similarly, in a database system an index for an attribute of a file is used to point to all the records in the file which contain a particular value for that attribute. In the EMPLOYEE file, for example, if an index existed for the attribute SAL, it would contain pointers to each record in the file with a salary value of, say, 10000.

Database indexes usually have a tree structure, with several "levels." Figure 6 shows a tree index for the salary attribute of the EMPLOYEE file, in thousands of dollars. In the third level each node of the tree consists of pairs of the form (v,a) of a value of the attribute and an address (i.e. location) of an "accession" list of records containing that value. The bottom row contains a value-address pair for each record in the file. Notice that

these nodes are sorted in order of the attribute's values.

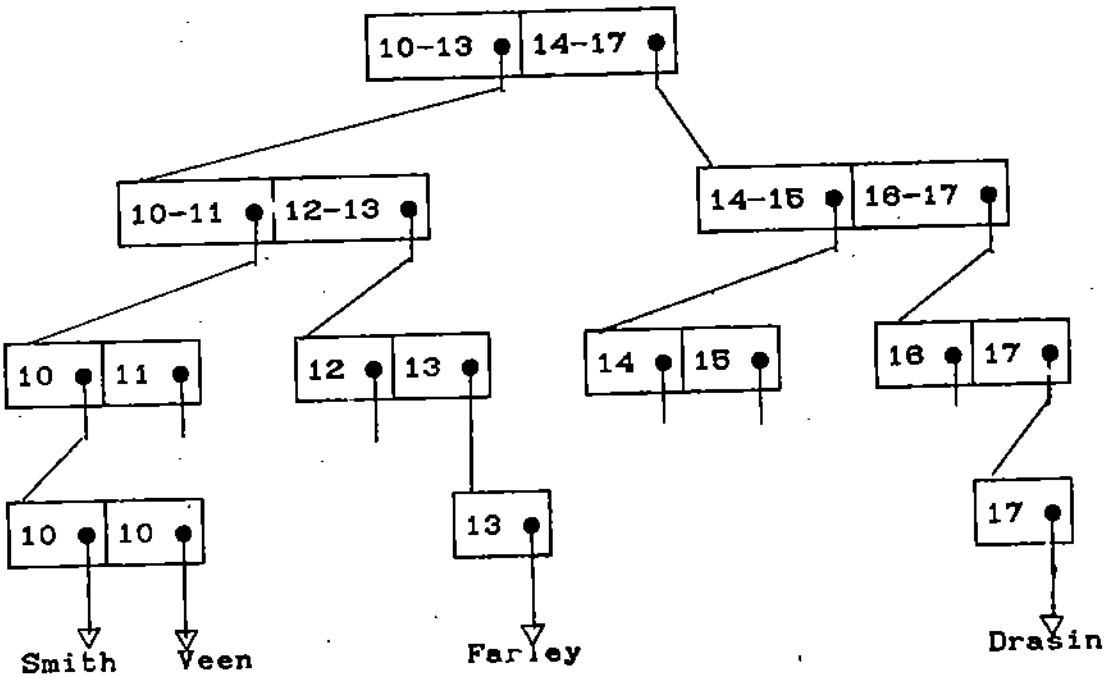


Figure 6. Tree index and accession list for salary attribute of EMPLOYEE file.

The higher level nodes of the tree also consist of value -address pairs. The value, though, may actually be a range of values, as shown in the first level of the index. The top level divides the file into broad categories which include many values of the attribute. Using the address pointers the tree is traversed from top to bottom, and the file is further subdivided. Finally, each leaf node corresponds to a single value. The degree of an index tree is the number of pointers which a higher level node

contains. We assume that the degree is constant for each node, resulting in a "balanced" tree. The height of the tree refers to the number of tree levels. Typically, three levels are sufficient for a database file.

In our access cost model we must include the cost of retrieving any indexes. For this purpose we assume, as is often the case, that each upper level index tree node occupies one physical page of storage.

The model uses indexes in two different ways. To perform the simple restriction SAL = 10000 we can begin at the top of the index tree and trace down through the levels until we find the set of pointers to records with SAL = 10000. On the other hand, if we wish to access the whole file of records we can use only the entire bottom row of the index to obtain pointers to each record. These pointers will be ordered according to the values of the indexed attribute. This characteristic is very convenient for the join operation.

LINKING. Quite frequently queries request data records which are logically related, such as a project record followed by records for all employees who may work on that project. Since all of these records may not be stored physically adjacent to each other, access time can be improved by linking them together via pointers, i.e., linked lists. The project (parent) record may contain a pointer to

one of its employee (child) records, which in turn would point to another employee (twin) record, and so on. In this way any hierarchical data structure can be implememted.

If, however, we have an employee working on more than one project we have a many-to-many relationship. We would then need an arbitrary number of pointers in the employee record to connect it with each of the appropriate departments. Those who are familiar with hierarchical or network database systems know that these many-to-many links cannot be efficiently implemented without the creation of additional record types [6]. Herin lies an advantage of the relational join operation over the network and hierarchical systems' counterparts.

CLUSTERING. Clustering is a storage method which takes records that may be retrieved together, and groups them together on a physical page of storage. This can reduce the number of page accesses required to retrieve the data. There are two types of clustering which we consider.

First, a single file, F, is said to be clustered with respect to attribute A if the pages on which F is stored can be ordered in such a way that the stored values of A appear in ascending order. Intuitively, this means that we first tape together all the pages of memory which contain records of file F. Then we sort the records of F in ascending order of attribute A. A file can clearly only be clustered with

respect to one of its attributes. If the file has an index, i, for that attribute A, then i is said to be a clustering index. Since an index also enumerates file F in increasing order of A, we see that by using a clustering index to access F, no page containing records of F will be accessed more than once. Figure 7 uses the employee file of figure 1 to illustrate a clustering index on the DEPT# attribute, and a non-clustering index on the SAL attribute.

Two or more files are said to be clustered with respect to each other if 1) a linking structure exists between them, and 2) the records are actually stored according to that structure, eg. in hierarchical sequence. To illustrate this we imagine a DEPARTMENT file which serves as the "parent" to the EMPLOYEE and PROJECT files of figure 1. The DEPT# attribute is then the link between each DEPARTMENT record and the EMPLOYEE and PROJECT records for that department. Figure 8 shows how these files could be physically clustered on the DEPT# attribute. Link searching, then, will generally require fewer page accesses if relations are clustered.

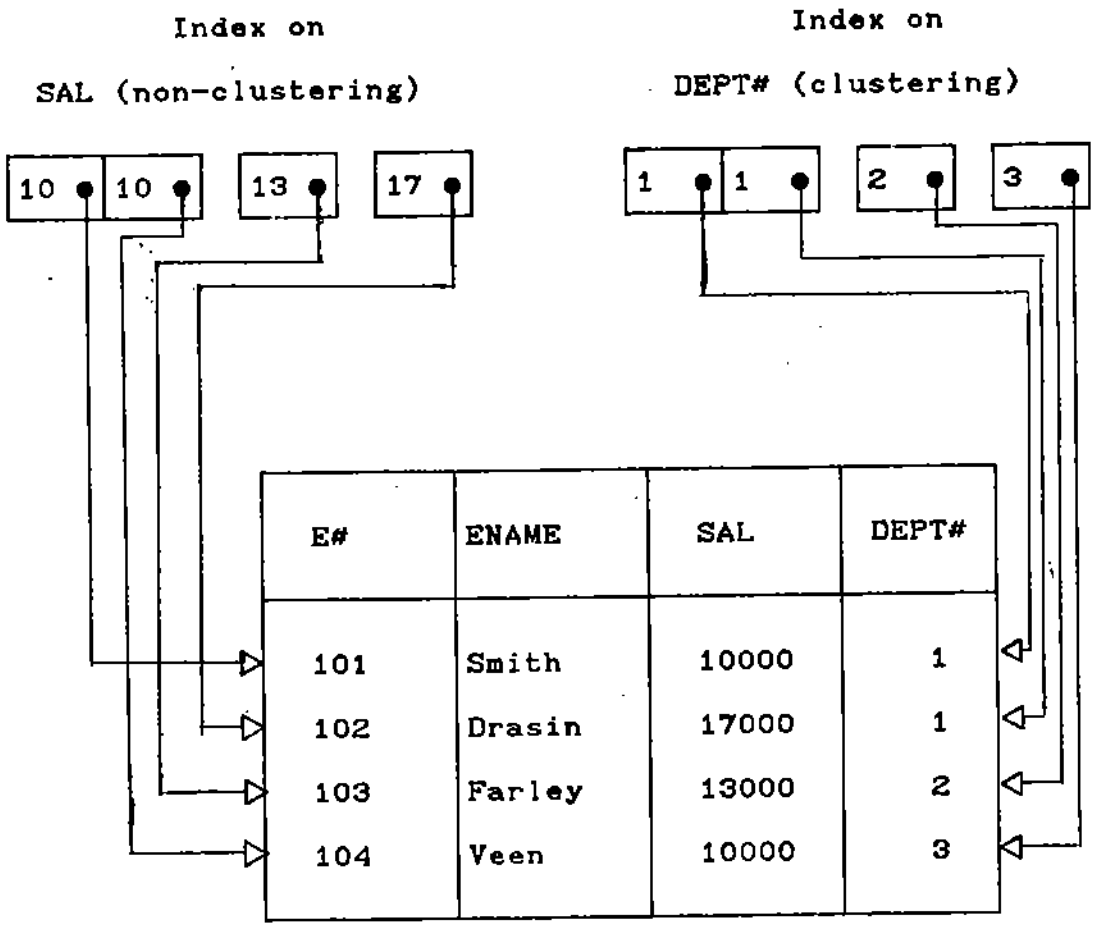


Figure 7. Clustering and non-clustering indexes for EMPLOYEE file.

Record	for	Dept# 1	
101	Smith	1000	1
102	Drasin	17000	1
P-1	1	May	
P-4	1	June	
Record	for	Dept# 2	
103	Farley	13000	2
Record	for	Dept# 3	
104	Veen	10000	3
P-2	3	April	
P-3	3	May	

Figure 8. Three files clustered hierarchically on DEPT# attribute.

G. ACCESS OPERATIONS.

The storage organizations mentioned tend to require a significant amount of maintenance work as records are inserted in and deleted from the database. Normally, then, these structures are built only in those places where heavy use is expected. Indexes, for example, will not be constructed for each attribute in a file, but only for those which are expected to be referenced often by queries. Because of this restraint, however, we can expect situations

in which the proper storage organization is not available when needed. For these occasions we provide the following two access operations.

SEQUENTIAL SCANNING.

The records from any file or cluster of files may be retrieved from the database in their physical sequential order. Each record may then be examined individually for a particular value for one of its attributes. Since the entire file or cluster of files is retrieved, this operation clearly involves a substantial number of secondary storage accesses.

SORTING.

At times during the data accessing process, it may be advantageous to have a set of records or pointers to records sorted on some attribute or on the record addresses. If the items being sorted can all fit into main memory, we will assume that an internal sort is used. If not, an external sort-merge with temporary intermediate files is required. For reasons of simplicity we implement an ordinary z-way sort-merge, "z" being a parameter of the system.

III. ACCESS PATH MODEL

We now describe the general model for access paths in a relational database system, based on the work by Yao [15]. In order to solve the general query presented in section II-E, the three operations of restriction, projection, and join must be performed on two files in the database. The order in which these operations are carried out is flexible. In addition, for each one of these operations the various storage and access techniques provide a variety of ways in which the operation can be accomplished. For example, the restriction "DEPT# = 1" which we earlier placed on the EMPLOYEE file could be accomplished in at least two ways. The file could be scanned sequentially, retrieving and examining each record to see if it passes the restriction. Alternatively, if an index exists for the LOCATION attribute we could use it to obtain pointers to those records which pass the restriction, and then retrieve only those qualifying records. It should be clear, then, that there are many possible access paths which will solve the query. The access path model can represent these various paths because it modularizes as much as possible the tasks to be performed. Searching an index is one module, for example, while performing a projection is another. These modules, or

components, are the heart of the model. Once defined, they can be linked together in any way that results in the proper evaluation of the query. In this way any one of the possible access paths may be modeled. In the remainder of this section we first describe each of the model's components and introduce the symbols which we will use to represent them. We then discuss assumptions made about the costs incurred by the components. Finally, we demonstrate ways in which they may be linked together in order to solve the general query.

A. MODEL COMPONENTS.

RI RESTRICTION INDEX. This component represents the task of performing a restriction operation with an index on the attribute being restricted. The index tree is traversed in order to obtain pointers to those records which pass the restriction, i.e., those which have specified value(s) restriction for the attribute.

FI FULL INDEX. Here any index which exists for the file is used. Instead of traversing the index tree from top to bottom, however, the entire terminal level of the tree is retrieved. This yields a list of pointers to each record n the file. Associated

with each record pointer is the value of the indexed attribute which the record has. By the nature of indexes, this list of pointers will be sorted on the values of the indexed attribute. Within each group of pointers corresponding to a particular value, the pointers will be secondarily sorted on the address of their corresponding records.

JI JOIN INDEX. This is a special case of the full index, where the index used is on the join attribute of the file, i.e., the attribute being used in the join operation. The resulting set of pointers and associated join attribute values will be sorted on the file's join values, greatly simplifying the join operation.

SP SORT POINTERS. A sort is performed on a list of pointers in order to simplify subsequent operations.

I INTERSECTION. Two lists of pointers to records in a file are intersected. Only pointers which are in both lists are kept.

DR DIRECT RETRIEVAL. Using a list of pointers to records of a file, this component represents the actual retrieval of those records, bringing them from secondary storage into main memory.

S SEQUENTIAL SCAN. Each record of a file is retrieved

by scanning every page of storage in the record cluster which contains records of the file.

SR SORT RECORDS. A sort is performed on the actual records from a file in order to simplify subsequent operations.

RF RESTRICTION FILTER. Examine the actual records from a file and filter out those which do not pass the restriction.

JF JOIN FILTER. Examine either the actual records from a file or record pointers which also indicate the join value of the records pointed to. Determine from these join values which records will participate in the join operation, and eliminate those which do not. This task involves communication with the other file being joined in order to determine which join values occur in both files. For this reason this task must be performed simultaneously on both of the files being joined.

X CARTESIAN PRODUCT. Given either records or pointers to records which participate in the join operation, perform the join. This is done by forming, for each join value, the Cartesian product of the corresponding records (or pointers) from one file with those from the other file. Since this task, too, involves interaction between the two files

being joined, it must be performed simultaneoulsy on both files.

FL FIND LINK. This component is used in order to obtain pointers to <u>each</u> record of a file by means of a linking structure from another file. The other file is searched by some means in order to find the link pointers.

'L" LINK SEARCH: ONE TO MANY. Trace the link structure from records of a parent file to records of the child file. Twin links in the child file must also be traced.

'L' LINK SEARCH: ONE TO ONE. In this special case of L there are no twin links to be searched.

P PROJECTION. Perform the desired projection on a record or set of records.

Pj PARTIAL PROJECTION. The projection which the query asks for may eliminate the join column of the files being joined. Obviously this may not be done until after the join operation has been performed. However, since projections reduce the files' sizes, it is advantageous to project as soon as possible, in order to reduce the space needed for any temporary storage of the files. This component represents a compromise solution. A partial

projection is done ahead of the join operation, but the join column is retained in the file. A total projection is performed later.

Prj PARTIAL PROJECTION. In this partial projection both the join and restriction columns of the file are retained for later operations.

B. COMPONENTS WITH "COSTS."

We have already mentioned that for our purposes "cost" is measured by the number of secondary storage accesses. We believe that this is a reasonable assumption, since, in terms of response time, secondary accesses are far more important than internal operations. In fact, many of the internal operations, such as join and projection, are common to all of the accesses paths, and so do not affect the comparitive path costs. The dominent factor in the differences in path costs is the number of secondary storage accesses.

On that basis, we go on to assume that all files and all indexes are initially in secondary storage. Hence any use of indexes, or any retrieval of records will result in secondary storage acccesses. This concerns the components:

RI FI JI DR S FL

Once records and pointers to records have been retrieved, we need to consider temporary storage. Since this involves more secondary accesses, our goal is to avoid it where ever possible. We assume that the processor employs a type of pipelining, where the records are processed to completion either individually or in small subsets, where ever possible. This avoids the costly intermediate manipualtion of entire files. This pipelined processing can be achieved with all of the above model components except the two dealing with sorting: SP and SR . The entire set of items being sorted must be processed together. If this set is too large to fit into main memory, then an external sort must be performed as described earlier in section II. G.

The results (output) of the access path processing are assumed to go directly to the user. Any associated costs are not within the scope of this model.

C. ACCESS PATHS

The specification of an access path to the data is a description of which components are to be used, and in what order they are to be applied. Obviously not every combination of components is sensible. We must determine which combinations will perform the required task: one restriction on each of two files, a join on the two files, and a projection. The set of access paths which we

illustrate here is not intended to include every possible path derivable from the model's components, but does include those paths which seem "reasonable."

The first thing to note in building an access path is that there are two files involved. The ordering of the components for one file need not be the same as for the other. Since the only time of interaction between the two occurs at the join operation (components JF and X), we specify only that the access paths for the files must be able to be synchronized at those points. For example, each row shown below could represent, from left to right, the order of operation on a file. The only stipulation is that records from both files which have the same join value must be able to enter the JF and X blocks simultaneously.

RI——DR——SR——JF——X——P

SS——RF——SR——JF——X——P

A second point to noice is that in performing the join operation, the two files can be treated either in parallel, or serially. In the first case the join fields of each file are examined together, in parallel, to determine by the corresponding join values which records are to be joined together. In the serial case there exists a parent-child link structure on the join fields of the two files. The join operation consists of searching one file for records

with a given value, then tracing their links to the second file. This property distinguishes cases 5 and 6 below from the first four.

1. Parallel Join. Cases 1 - 4.

In this group the access paths for the two files are essentially independent of each other , outside of the synchronization during the join operation. Thus all the paths shown here are for a single file. Any two such paths may be chosen to solve the general query: one for the first file, and another for the second.

We said earlier that the costly components of the model are indexes, record retrievals, and record sorts. Record sorts are always performed as immediate preparation for the join operation, and indexes are used in connection with restrictions, joins, and/or record retrieval. Thus the three general operations which may incur costs are restriction,join, and record retrieval. These operations can be done in any order, and the four categories of possible access paths which follow reflect the six permutations of these three operations.

CASE 1: JOIN - RETRIEVAL - RESTRICTION.

JI — JF — DR — RF — X — P

This access path begins by using the terminal level of an index on the join field to obtain record identifiers (p,v) consisting of pointers to the records, along with their associated join values. The join filter is then applied to screen out pointers to those records which do not participate in the join. The remaining records are then retrieved and examined to see if they pass the restriction on that file. Those that succeed move on to the cross product operation. Finally the proper columns are projected out. Only the JI and DR components involve secondary storage access. Thus re-arrangements of the remaining components do not alter the access path cost. Two such variations are:

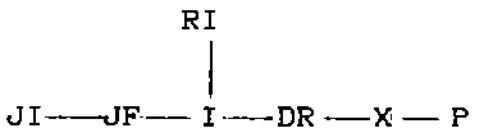
JI—JF—DR—RF—Pr—X—P

JI—JF—DR—Prj—RF—X—P

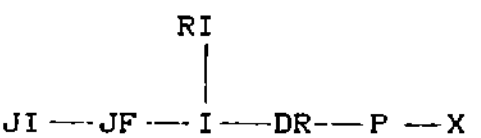
In these variations partial projections are performed ahead of the cross product and restricton filter operations. The remaining projection is performed last.

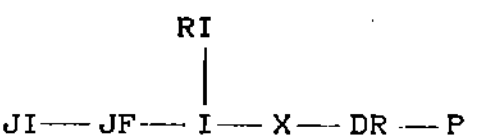
If this access path is used on both files involved in the query, the cost may be reduced slightly by allowing feedback from one path to the other. If all records from one file having a particular join value fail to pass the restriction filter, then no records from the second file with that join value need to be retrieved. Feedback of such information could reduce the path cost of the second file.

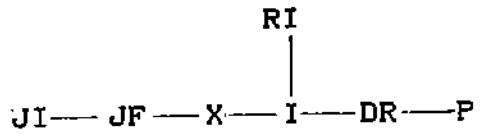
CASE 2: JOIN - RESTRICTION - RETRIEVAL AND RESTRICTION - JOIN - RETRIEVAL.



This path uses indexes on both the join field and the restriction field. The restriction index obtains pointers to those records which pass the restriction. The join index and join filter find pointers to those records which participate in the join. Intersection of these two pointer sets is performed easily, since the restriction list is sorted primarily on record address, and the join list is sorted secondarily on record address. After intersection these lists of pointers, the qualifying records are retrieved, joined with those from the other file, and projected. Components involving secondary storage accesses are JI RI and DR . Variations of this access path which have the same costs are:







These variations merely perform the actual cross product part of the join operation at different times.

CASE 3: RESTRICTION - RETRIEVAL - JOIN.

A) RI — DR — SR — JF — X — P

In this access path a restriction index and direct retrieval are used to obtain the records which pass the restriction. The sort is needed since the join process requires that the records be sorted on the join values. The sort may be internal or external, depending on the size of the restricted file. Access paths of identical costs are:

RI — DR — SR — JF — Pj — X — P

RI — DR — SR — Pj — JF — X — P

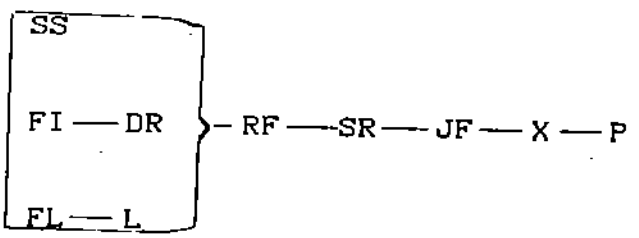
B) If the record sort is external, its cost could be reduced by applying the partial projection ahead of the sort. This would result in less temporary storage space needed during the sort. The access path is then:

RI — DR — Pj — SR — JF — X — P

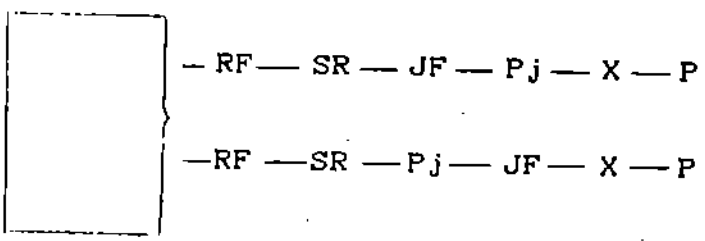
CASE 4: RETRIEVAL- RESTRICTION - JOIN AND RETRIEVAL - JOIN - RESTRICTION.

In the following access paths the first step is to retrieve the file records from storage. This can be accomplished by a sequential scan, by use of the entire terminal level of some index followed by a direct retrieval, or by a find link and trace link from a parent file. To avoid unnecessary repetition in the access paths shown, we group these three alternatives together into one box. After the records have been retrieved they are checked for the restriction and then sorted on the join value. The records are then joined with the other file, and projected.

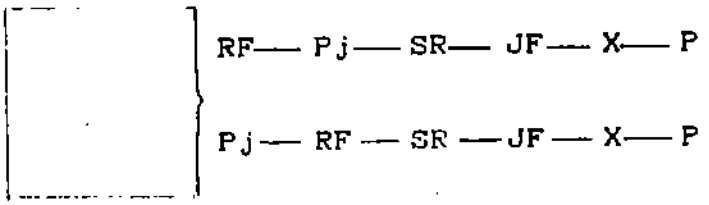
A)



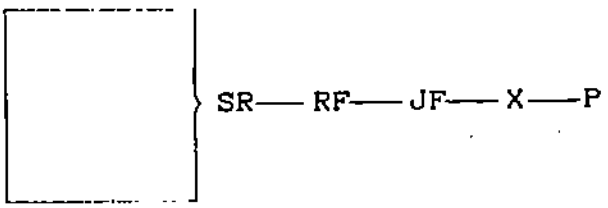
Variations with identical costs are:



B) If the sorting is external then costs may be saved by moving the partial projection ahead of the sort:

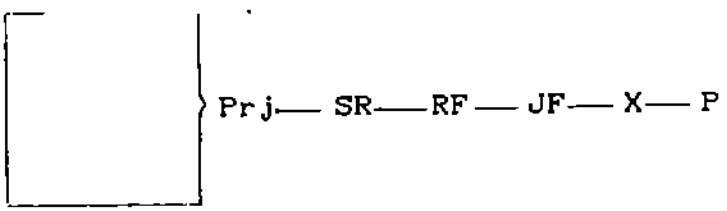


C) The restriction filter may be moved in back of the sort. Now the sort is on the entire file, so secondary storage is probably needed during the sort operation.

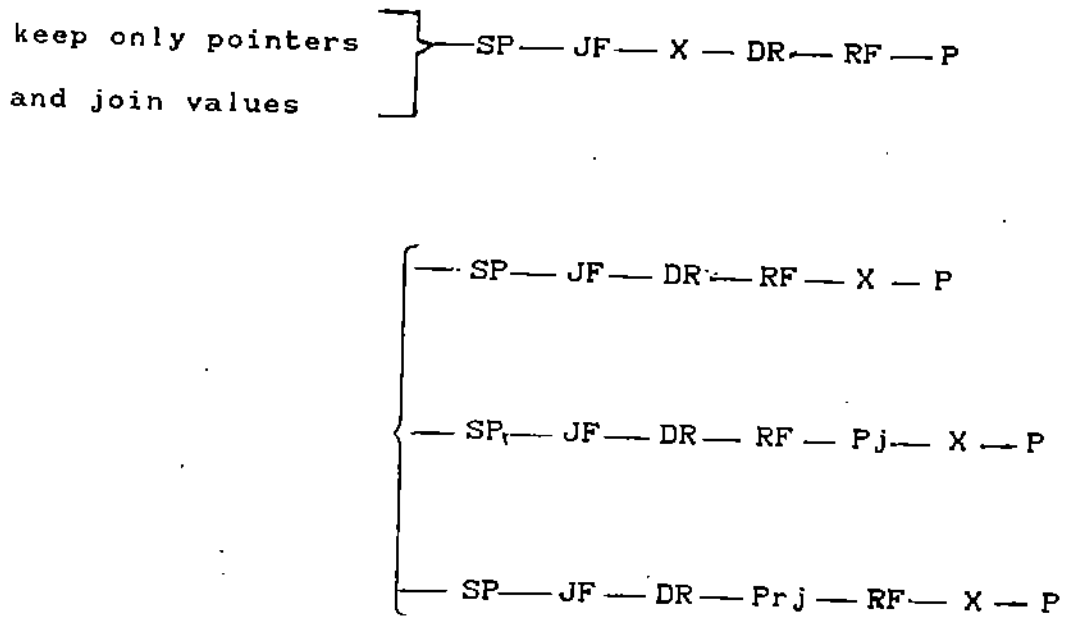


Permutations of the operations following the sort will have no effect on the path cost, since no temporary storage is involved.

D) External sort oosts may again be reduced by moving the partial projection ahead of the sort:



E) A final variation of this case performs an internal sort. This can be done if, after a record is retrieved, the join value is kept along with a pointer to the record. The rest of the record is then dropped, and the sort is done only with pointers. The remainder of the record is retrieved again later. The following paths would have the same cost:

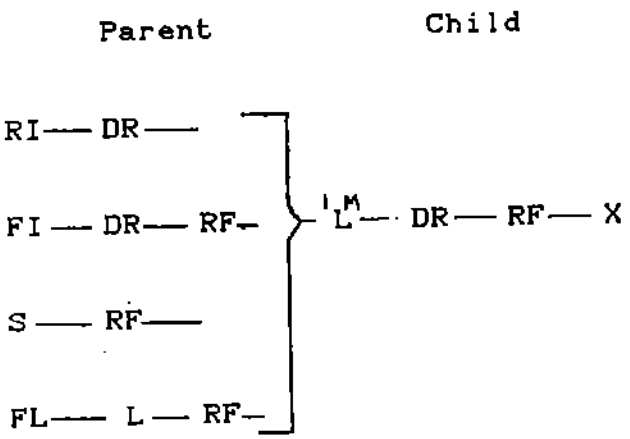


2. Serial Join. Cases 5 - 6.

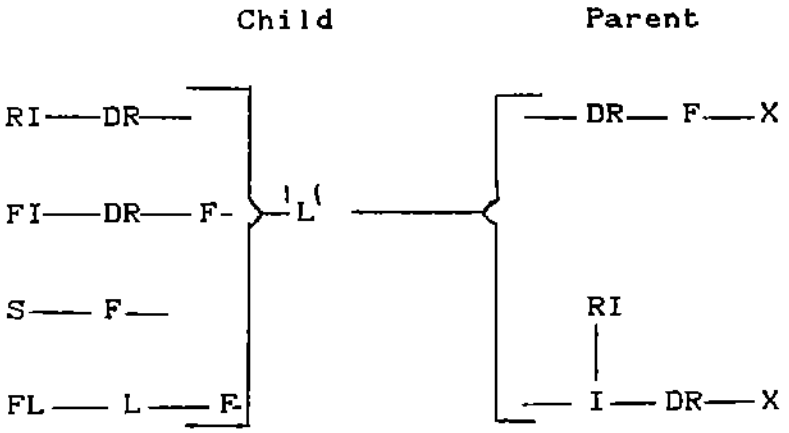
In this group of access paths the two files are assumed to have a one-to-many linking relationship on the join attributes (a one-to-one relationship is a special case).

CASE 5: PARENT - CHILD. Here the parent record in the first file is linked to its children having the same join value in the second file. The parent file records are accessed and restrictd by one of several available methods. The links are obtained from those records which pass the restriction. Joining occurs during the process of tracing the links to the child file, obtaining and restriction the child's records, and tracing to their twin links. We assume that each record in the child file is linked by either a parent-child or child-twin pointer, that is, there are no "orphans." The cost of this case will depend upon which method is specified for accessing the parent file. There

are four options for the parent search.



CASE 6: CHILD - PARENT. Here each record in the child file has a link pointing to its parent record in the parent file. It is assumed that each child, in fact, has a parent (with the same join value). The child file is searched and restricted first. From those records which pass, the link is obtained and traced to the parent file. The parent record is then obtained, restricted, and joined to the child. An alternative method uses a restriction index on the parent file. The restriction on the parent is then applied by checking each incoming link pointer with the list of pointers obtained from the index. In this way only those parent records which pass the restriction need to be retrieved.


Child
Parent
RI—DR—
FI—DR—F-
S—F—
FL—L—F-
L'
—DR—F—X
RI
—I—DR—X

IV. COST EQUATIONS

Having described the access model components, and showed ways in which they may be connected, we now present a series of cost equations for those access paths. These equations enable us to compare the performance of the access paths under various circumstances. Included in the parameters for the equations are characteristics for each of the three major factors of database performance: the type of query, the physical organization of the data, and the logical relationships among the data. After listing these parameters we proceed to examine the costs of those components of the model which incurr costs under our assumptions. The access path costs then, are found basically by summing the individual component costs.

A. PARAMETERS.

MEM	Number of pages in main memory
BL	Number of pages per block, used in external sorting
Z	Z-way sort used in external sorting

S	Size, in logical records, of a file
PR	Number of logical records of a file per page of memory
NLEV	Number of levels in an index for an attribute of a file
PI	Number of pointers in each index page
SCLUS	Size in pages of the cluster containing a file
FAN	Parent-child fan out for files logically linked
RVAL	Number of values for an attribute being restricted
JVAL	Number of values for an attribute being joined
RSEL	Selectivity of a restriction placed on a file i.e., percentage of the file passing the restriction
JSEL	Selectivity of a join operation i.e, percentage of the file participating in the join
P	Selectivity of a projection i.e, percentage of the file remaining after a projection
Pj	Selectivity of a partial projection which includes the join attribute

Prj — Selectivity of a partial projection which includes the restriction and join attributes

PGS(F,G) — Number of pages accessed in traversing from a record in file F to te next record in file G (maximum = 1.0)

Many of these parameters may either be repeatedly specified: for each file and file attribute; or, for simplicity, may be given a single "average" value. The accuracy as well as the complexity of the cost evaluation is somewhat dependent on which alternative is chosen. The last parameter, PGS(F,G), is actually a function of several of the other parameters. It's derivation appears in Appendix A.

B. COMPONENT COSTS

RI Restriction Index.

For a simple restriction of file F, one page from each upper level of the index must be retrieved, for a total of NLEV pages. The number of pages containing pointers to records satisfying the restriction is given by (S(F)*RSEL)/PI.
Total Cost = NLEV + [S(F)*RSEL]/PI.

JI , FI Join Index or Full Index.

If an index is used to obtain pointers to each record of a file F, then only the terminal level of the index needs to

be retrieved. The number of such pages is S(F)/PI.

DR Direct Retrieval.

This component takes a list of (sorted) record pointers, such as from an index, and retrieves the corresponding records from the file F. The number of pages accessed in retrieving the records is affected by whether or not the records to be retrieved are consecutive records of the file.

NON-CLUSTERING LIST. Here K records are chosen at random from SCLUS(F) pages containing a total of SCLUS*PR records. Yao [14] has derived an exact solution for the number, P, of pages accessed:

$$P = SCLUS(F)\left[1-\prod_{i=1}^{K}(SCLUS(F)*PR*d-i+1)/(SCLUS(F)*PR-i+1)\right],$$

where d = i - 1/SCLUS(F)

In Appendix B we show that a good approximation of P which is somewhat easier to calculate is:

$$P' = SCLUS(F)\left[1-((SCLUS(F)*PR*d+1)/(SCLUS(F)*PR+1))^{\frac{K}{m}}\right]$$

CLUSTERING LIST. Here N consecutive records of file F are retrieved at a cost of N*PGS(F,F).

COMBINATION. The pointer list could result from the intersection of record pointers from clustering and non-clustering indexes. In this case the clustering list has effectively reduced the file size to, say, X*SCLUS(F) consecutive pages, where $0<X\leq 1$. The non-clustering list

then chooses K records at random from these remaining pages. The resulting cost is identical to that of a non-clustering list, if we replace SCLUS(F) with X*SCLUS(F).

S Sequential Scan.

Since a sequential scan examines each page of the cluster containing file F, its cost is simply SCLUS(F).

SR Sort Records.

If the items to be sorted will fit in main memory, then an internal sort can be performed, with no pages of secondary memory being accessed. When an internal sort of a file of M pages is required, the pages are divided up into B = M/BL blocks. Using a Z-way sort merge, the number of passes required to sort the file is $\log_Z(B)$.[3] If for each pass each block must be read and then written again, we would have a cost of $2*B*\log_Z B$ for the sort. However, the first pass on each block can be performed while the items are still in main memory from the previous access path component, eliminating the initial read. Also, after the last pass the records can be passed on to the next component, eliminating the final write. Thus the sort cost, in block accesses, is $2*B*[\log_Z(B)-1]$, or

$$SR(M) = \begin{cases} 2*(M/BL)*[\log_Z(M/BL)-1]) & \text{if } M>MEM, \\ 0 & \text{otherwise.} \end{cases}$$

L DR Parent-Child Link Search.

We have a linking structure between a parent file, P, and a child file, C. We let N be the number of link pointers in the parent file which must be traced and the corresponding child records retrieved. For each such child the fanout parameter, FAN, indicates the number of twin records which must also be retrieved. The cost of this link searching depends on the physical arangement of the two files.

P, C NOT CLUSTERED. Files P and C are not on the same pages of memory, so one new page must be accessed for each parent-child link traced, for a total of N accesses.

Twin Clustering. If the linked list of twins is a clustering list, then each link points to the very next child record in the database. FAN-1 twins must be retrieved for each parent-child link, at a cost of TW = (FAN-1)*PGS(C,C).

No Twin Clustering. Here a twin pointer may point to any record of the cluster which contains file C. Thus the probability of accessing a new page is [SCLUS(C)-1]/SCLUS(C). The number of page accesses in tracing the twin linked list is then expected to be TW = (FAN-1)*[SCLUS(C)-1]/SCLUS(C).

The total cost of the parent-child link search is then N*[1+TW].

P, C Clustered.

Files P and C are phisically clustered in hierarchical order, according to the values of a particular attribute. For example, a company's file system might have a PLANT record followed by each EMPLOYEE record whose LOCATION attribute has the same value as that in the PLANT record.

If the linking structure is on the same attribute as the physical clustering, then the parent-child link always points to the very next child record in the cluster. Similarly, each twin link points to the very next child record. Thus the total cost is

N*[PGS(P,C) + (FAN-1)*PGS(C,C)]

If the linking structure is not on the same attribute as the physical clustering, then each link could point to any page in the physical cluster containing P and C. is (SCLUS(C)-1)/SCLUS(C). The total cost of the parent-child link search is then

N*FAN*(SCLUS(C)-1)/SCLUS(C).

L DR Link Search: one to one.

We again have a linking structure between a parent file, P, and a child file, C. This time, however, a record in file P has multiple child pointers; that is, a parent record has a link to each of its child records. If the parent-child fanout is 1, then this is a special case of the one to many link search described above, with no twin pointers.

Otherwise, further cost analysis is required. We let N again be the number of parent-child links which must be traced.

P, C NOT CLUSTERED. In this case files P and C are not on the same pages of memory, so one page access is required for each link traced. The total cost is N.

P, C CLUSTERED. In this situation files P and C are clustered in hierarchical sequence, according to the values of a particular attribute.

If the logical link is on this same attribute, then it connects the parent either to the immediately following record of the child file, or to one of its successive twins. On the average, the desired child record will lie half way down the twin linked list. Thus the total distance, in pages, from parent to child is, on the average,
min[1,PGS(P,C) + ((FAN-1)/2)*PGS(C,C)].
For all such links the total cost is then
N*min[1,PGS(P,C) + ((FAN-1)/2)*PGS(C,C)].

If the linking structure is <u>not</u> on the same attribute as the physical clustering, then we again access a new page with probability [SCLUS(C)-1]/SCLUS(C). With N such links the total cost is
N*[SCLUS(C)-1]/SCLUS(C).

C. ACCESS PATH COSTS.

We finally take a closer look at the complete access paths described earlier, and try now to express the cost for each alternative. Recall that Cases 1-4 deal with a single file only. Cases 5 and 6 take into consideration both parent and child files.

CASE 1.

NO FEEDBACK. With no feedback we retrieve each record in a file, F, which participates in the join operation.

The cost for JI is S(F)/PI.

Let C = $\begin{cases} 1 & \text{if JI is a clustering index} \\ 0 & \text{otherwise.} \end{cases}$

The number of records to be retrieved is N = S(F)*JSEL.

Then the cost for DR is

C*[N*PGS(F,F(] + (1-C)*[P'(SCLUS(F)*PR,SCLUS(F),N)], where P' is the direct retrieval function derived earlier.

Total cost is then

S(F)/PI + C*[N*PGS(F,F)] + (1-C)*[P'(SCLUS(F)*PR,SCLUS(F),N)]

FEEDBACK. If file F is being joined with file G, we assume that in this case the restriction has already been placed on G. This may eliminate all records in G with a given join value, v, and so no records in F with join value v need be retrieved. In order to find the number of join values of G which survive the restriction we observe the following:

Mg = JVAL*JSEL is the number of join values in G which would normally participate in the join operation.

Ng = S(G)*JSEL is the number of records in G which would normally participate in the join operation. From these the restriction selects only Kg = Ng*RSEL records. If we picture the Ng records of G as being sorted by join value, then we can imagine Mg groups, or "pages," of records with each group corresponding to one join value. The restriction operation randomly selects records from these "pages," and we wish to know how many "pages" have been "accessed" by the restriction. The cost equations presented for the direct retrieval operation can be applied to this analogous situation, and we then find that P'(Ng,Mg,Kg) gives us a good approximation to the number of join values which remain. File F then need only retrieve records which have one of those remaining join values, in other words, (S(F)/JVAL)*P'(Ng,Mg,Kg) records.

Thus in the cost equation for the feedback path, we substitute N = (S(F)/JVAL)*P'(Ng,Mg,Kg)).

CASE 2.

Given a file, F, the cost for JI is S(F)/PI.

The cost for RI is NLEV + S(F)*RSEL/PI.

After the intersection of the two index pointer lists the number of records remaining to be retrieved is

K = S(F)*RSEL*JSEL.

SCLUS(F) if neither index is clustering

Let M = { SCLUS(F)*JSEL if the join index is clustering
        { SCLUS(F)*RSEL if the restriction index is clustering

and N = M*PR

Then the direct retrieval cost is P'(N,M,K).

Total cost for this case is then:

NLEV + (1+RSEL)*S(F)/PI + P'(N,M,K)

CASE 3.

PATH A. For a file, F, the cost for RI is NLEV + S(F)*RSEL/PI.

The number of records to be retrieved is K = S(F)*RSEL.

Let M = { SCLUS(S) if the restriction index is not clustering
        { SCLUS(S)*RSEL if it is clustering,

and N = M*PR = the number of records in those M pages.

Then the direct retrieval cost is P'(N,M,K).

The cost of sorting the resulting K/PR pages of records is SORT(K/PR).

Thus the total cost for this path is:

NLEV + S(F)/PI + P'(N,M,K) + SORT(K/PR)

PATH B: PARTIAL PROJECTION AHEAD OF SORT. The size of the file to be sorted is reduced by a partial projection to (K/PR)*Pj pages.

Total cost for the path is then:

NLEV + S(F)/PI + P'(N,M,K) + SORT((K/PR)*Pj).

We see that the cost of path B will always be less than or equal to that of path A.

CASE 4.

One of the following types of record retrieval must be specified by the user for this case in order to obtain all the records from the file, F.

FI DR FULL INDEX AND RETRIEVAL. The cost of FI is S(F)/PR. The whole file is to be retrieved, so the number of records is K = S(F). The number of pages on which those records are stored is M = SCLUS(F), and the total number of records on those pages is N = SCLUS(F)*PR. The total retrieval cost is then:
R = S(F)/PI + P'(N,M,K).

S SEQUENTIAL SCAN. If this option is chosen, the retrieval cost is simply R = SCLUS(F).

FL L TRACE LINK. If this option is specified the parent file, P, of F must be searched using either of methods (1), or (2) above (or even (3), recursively). Let this cost be represented by R(P). Links are then used to access file F. The cost, R(C), of tracing the links and retrieving the child records is calculated by the L DR or the L DR component. Parameters needed for this calculation include whether or not the files are clustered on the linking attribute, or even clustered at all; and whether or not the twin linked list (if any exists) is clustering. The total retrieval cost of FL is then

R = R(P) + R(C).

Having chosen a retrieval path for F, there are five alternative paths for completing the query, as described earlier.

PATH A. Restrict the retrieved file and sort them for the join operation.
Sorting cost is S = SORT(S(F)*RSEL).

PATH B. Partial projection ahead of sort.
Sorting cost is S = SORT(S(F)*RSEL*Pj).

PATH C. Sort file before restriction.
Sorting cost is S = SORT(S(F)).

PATH D. Partial projection ahead of sort, ahead of restriction.
Sorting cost is S = SORT(S(F)*Prj).

The total cost for each of these four paths is then R+S. Based on the assumptions of our cost analysis, it is clear that path B is the most economical access path of the four alternatives.

PATH E. Keep pointers only.
Since this last alternative is somewhat different from the others, we consider it separately. The items being sorted here are not records, but only pointers to the restricted records, along with their join values. The number of

pointers is S(F)*RSEL. These fit onto S(F)*RSEL/PI pages of memory, so the sort cost is S = SORT(S(F)*RSEL/PI). After the join operation we must again retrieve the records. There are K = S(F)*RSEL*JSEL records to be retrieved from M = SCLUS(F) pages, containing a total of N = SCLUS(F)*PR records. So the direct retrieval cost is P'(K,M,N).

Total cost for this path is

R + S + P'(K,M,N).

CASE 5 PARENT-CHILD.

The method for carrying out the task of retrieving and restricting the parent file is specified by the user. Since these costs have been explained above, we will denote the cost of the chosen path by R(P). We must now determine how many links there are between the two files. We have assumed that there are no "orphan" children, and since the linking structure is on the join attribute, we see that all of the child records participate in the join operation. This is not necessarily true for the parent file. The number of parent-child links is then equal to the number of join values in the child file, JVAL. The parent file, however, has been restricted before this linking operation takes place. This reduces the number of links which need to be traced to N = JVAL*RSEL. Given N, the component L DR is then able to calculate the linking cost, L, based on the physical clustering of the two files.

Total cost for this case is then R(P) + L.

CASE 6 CHILD-PARENT.

Here the first task is to retrieve and restrict the child file. The user must specify which retrieval method to use, and since, again, we have covered the various retrieval options above, we simply state the cost as R(C). We have assumed that each child has a parent record with the same join value. Thus the number of children participating in the linking (i.e. in the join) operation is the number of children passing the restriction: N = S(C)*RSEL.

PATH A.

In the first alternative each link to the parent file is traced, and the parent records are retrieved and restricted. Given N, the component L DR is able to calculate the linking cost, L, based on the physical clustering of the two files.

Total cost for this path is then R(P) + L.

PATH B.

For the second alternative the number of links to be traced is reduced by intersecting the set of link pointers with the list of pointers obtained by a restriction index on the parent file. The number of remaining links is then

N = [S(C)*RSEL(child)]*RSEL(parent).

Component L DR is then able to compute the linking cost, L, based on the physical clustering of the two files.

The added index cost of component RI is

NLEV + S(P)*RSEL/PI.

Total cost for this path is then

R(P) + NLEV + S(P)*RSEL/PI.

V. TEST RESULTS

The cost equations just described have been implemented and tested on the dual CDC 6500 computer at Purdue University. The program consists of approximately 300 lines of FORTRAN IV code. Both the parameters for the model and the cases to be tested are specified in the program's main procedure, which is the only portion of the program needing compilation with each run. Typical runs which compute and roughly plot 50 points of the access costs of up to four different paths consume less than three seconds of CPU time.

A. COMPARISONS WITH PREVIOUS WORKS.

In this first set of results we compare access path costs of the model with several previous works in the same area. The purposes of these comparisons are to 1) indicate the validity of the model by showing results which are similar to those found earlier, and 2) illuminate those areas in which the model is an improvement over earlier work.

Blasgen, et al.

We first compare our work with that of Blasgen [3], whose approach to path analysis is very similar to ours. Of the ten access paths analysed in that paper we look here at two which can be easily compared with paths in our model. The three access paths of theirs are essentially the same as ours, but we will show that differing cost assumptions can result in differing cost analysis of the paths.

CASE 4-C: RETRIEVAL-RESTRICTION-JOIN. Our analysis of this access method is very similar to that of Method 4 found in the Blasgen paper. The path under consideration uses a clustering full index on each file to retrieve the records, and a partial projection is performed before the sort:

both files: FI——RF——Pj——SR——JF——X——P

Based on the cost equations given in their paper we compare their analysis of this access path with ours. The only differences appear in the evaluation of sorting costs. First, we obtain greater flexibility by allowing for an internal sort if the record file is small enough to fit into main memory. When the files become large enough to require an external sort, there is a marked increase in the access cost. The second difference in analyses concerns how the records are handled after the last pass of the sort has taken place. The Blasgen equations indicate that the sorted records must be written onto a temporary file before

proceeding with the join operation. We assume that the sorted records are pipelined, one join value at a time, directly from the last pass of the sort to the join operation. This results in the saving of an extra write to and subsequent read from secondary storage of the sorted record file. To illustrate these differences, Figure 9 shows a plot of access cost as a function of file size for both methods. The two sets of curves correspond to two restriction selectivity values, 1.0 and 0.1. Other parameters were chosen to be compatable with the Blasgen analysis. These parameter values, and those for the following graphical results are shown in Tables 1 and 2. As one would expect, the difference in sorting costs is most apparent when many records are being sorted, which happens when the file sizes are large, and when no records are filtered out by the restriction.

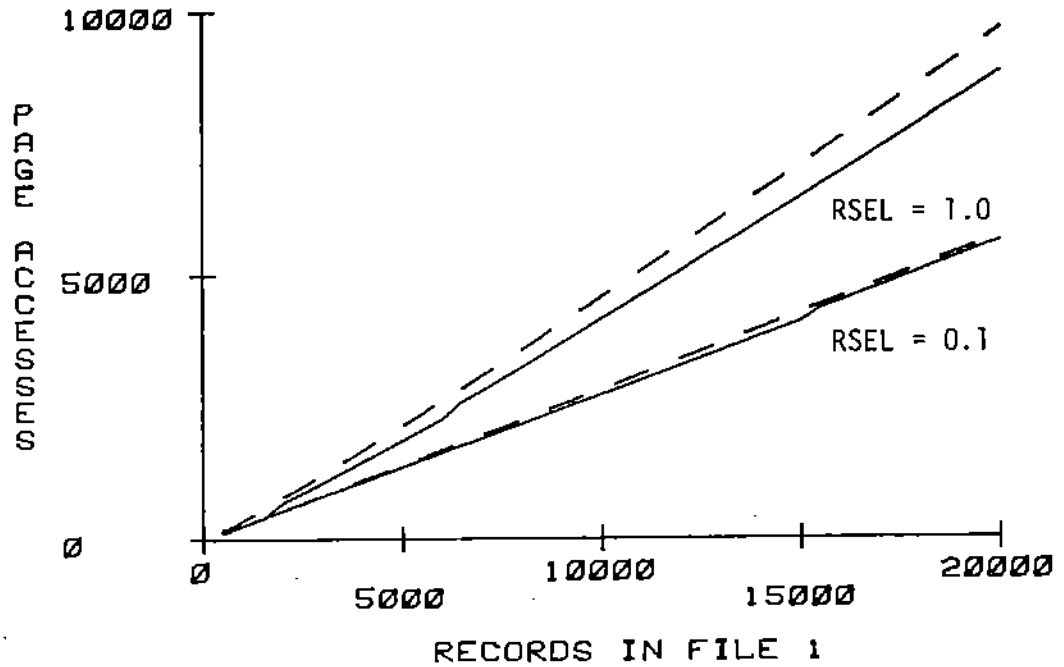


Figure 9. Case 4-C (solid) and Blasgen (dashed).

CASE 1-B: JOIN-RETRIEVAL-RESTRICTION. We now examine our Case 1 and compare it with Method 1 in the Blasgen paper. The access path for each file consists of a join index, direct retrieval, restriction filter, cross product, and projection on each file. Both analyses allow for feedback from the first file to the second. Recall that this means that if all records in the first file with a given join value fail to pass the restriction, then no records in the second file with that join value need even be retrieved. We have used the function P', as derived in Appendix 1, to approximate the number of join values which remain to be retrieved. The Blasgen equations, however, implicitly

assume a linear relationship between the restriction selectivity and the number of surviving join values: a p% selectivity retains p% of the join values. But this is true only when each record has a unique join value, that is, only when a one to one or a one to many relationship holds between the first file and the second. Many to many relationships render the equations inaccurate. For comparitive purposes we first model the Blasgen situation as closely as possible, by setting our parameter JVAL to the number of records in file one. In this case the Blasgen equations are correct. Also, in this situation our approximation function P' suffers its largest error (see Appendix 1). Figure 10 compares the two methods for two values of RSEL, and indicates that this error is significant only in the worst case for P': JVAL = file size, RSEL = 1.0. The access paths here are utilizing a clustering join index for the retrieval process.

Figure 11 illustrates the significance of the JVAL parameter, which previous cost models lack. By choosing RSEL = 0.1 we assure the accuracy of the function P'. We see that as the first file's value of JVAL decreases, the cost of access path 1-B increases dramatically. When a larger number of records possess the same join value, a smaller percentage of join values will be entirely eliminated by the restriction on the first file. Thus the feedback is less effective, and retrieval costs for the

second file increase. In fact, as JVAL decreases the cost of path 1-B approaches that of 1-A, where no feedback is allowed.

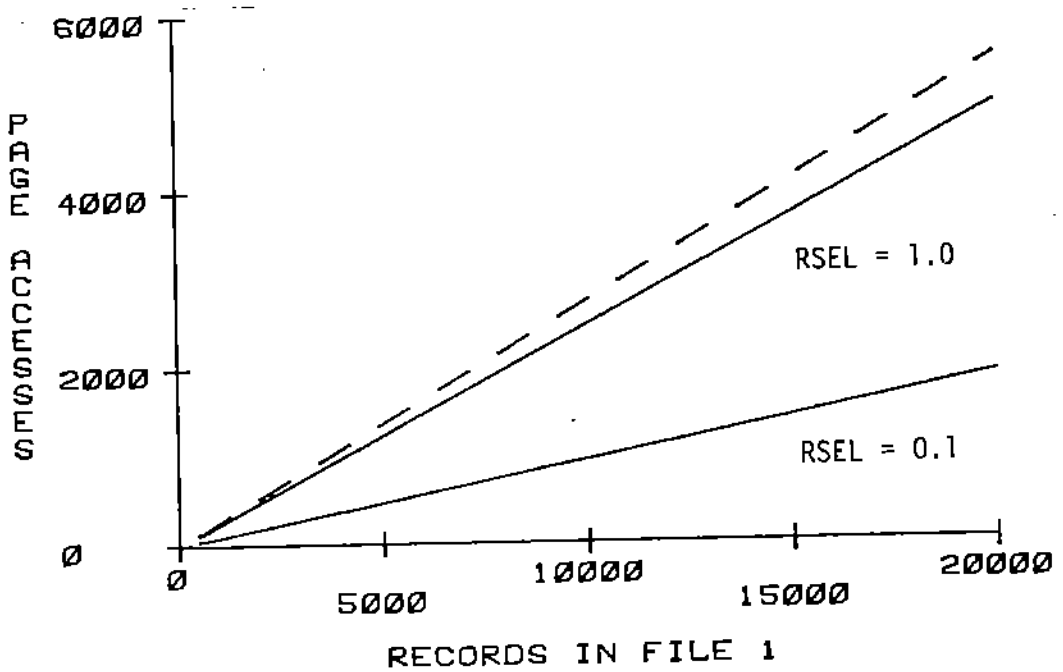


Figure 10. Case 1-B (solid) and Blasgen (dashed).

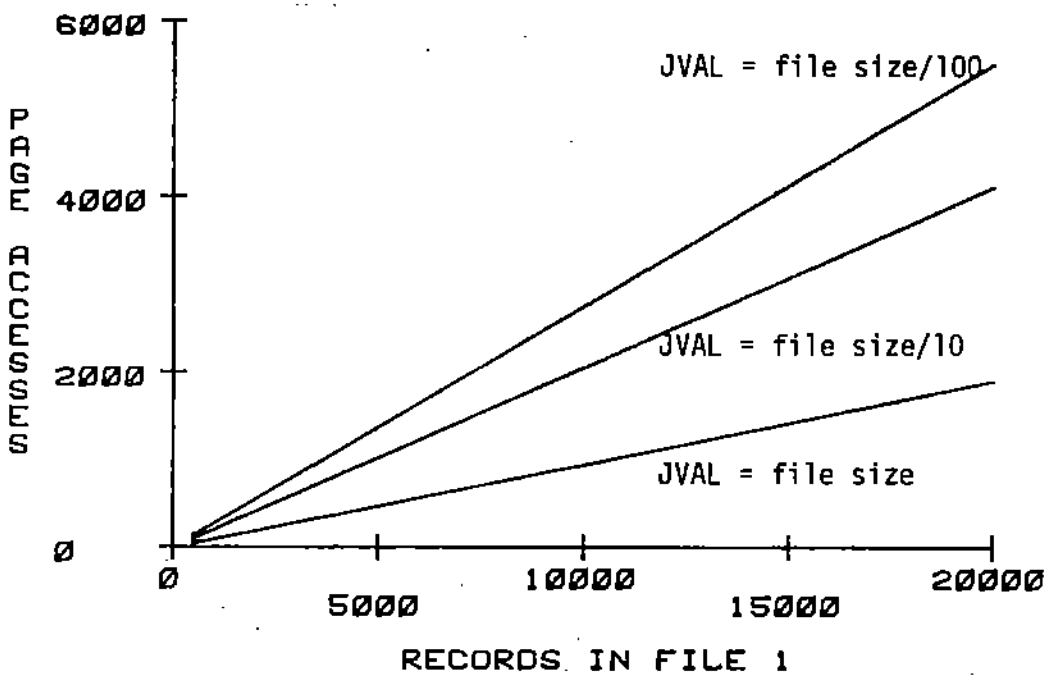


Figure 11. Case 1-B (solid) and Blasgen (dashed).

Astrahan and Chamberlin.

We now go on to make a similar comparison of our model with the access paths described by Astrahan, et al., in their work on implementing the SEQUEL query language [1]. We look at an example of our general relational query and show how it could be written in SEQUEL. We then present the access algorithm which they propose for solving this query, and, under similar cost assumptions, compare its performance with our corresponding access path.

Recall the example given earlier of a general query on the files EMPLOYEE and PROJECT: "Print the names and project id's of all employees whose salary is 10,000, who are

working in departments which have projects due in May." In SEQUEL this query can by written in the following nested form:

```
Select NAME, V
    from E in EMPLOYEE
    where SAL = 10,000
    compute V
        select PROJ-ID from PROJECT
        where MONTH-DUE = "May"
        and DEPT# = E.DEPT#
```

For a query of this type, Astrahan uses the following access path algorithm:

Step 1. Perform the restriction on the employee file to determine the set E_1 of records containing SAL = 10,000.

Step 2. Perform the restriction on the PROJECT file to determine the set E_2 of records containing MONTH-DUE = "May."

Step 3. For an employee record passing the restriction, find its department number.

Step 4. Determine via an index the set P_3 of records in PROJECT which have that same department number.

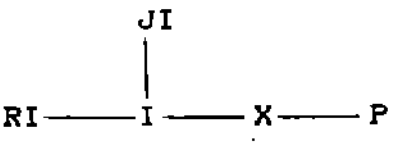
Step 5. Intersect the sets P_2 and P_3, and retrieve the resulting set of records.

Step 6. Print out the employee name and corresponding project-id's.

Step 7. Repeat steps 3-6 for each employee record in set E_1.

If we examine the cases where indexes exist on both the restriction and join attributes, we see that this algorithm can be described with these access path modules:

EMPLOYEE file RI—— DR —— JF —— X —— P



This corresponds very closely to cases 3 and 2, respectively. The major difference is that the employee file records passing the restriction are not sorted on the join values before the join operation occurs, as would happen in Case 3.

Without this added sorting operation the Astrahan algorithm can result in unnecessary record retrievals. As each restricted employee record is retrieved its DEPT# value is obtained and input to the DEPT# index (JI) on the PROJECT file. Those project records having the same department numbers and passing the due date restriction are then retrieved and joined with the employee record. Redundancy can occur if several qualifiying records have the same department number, say 1 (i.e., with a many-to-many relationship between EMPLOYEE and PROJECT). These employee records each need to be joined with the identical set of

project records. But these employee records will not normally be retrieved consecutively since the retrieval order is based on the salary attribute, not on the department number attribute. This results in re-retrieving the project records for department 1 for each employee in the department.

These redundant retrievals can be eliminated if, as in Case 3, the qualifying EMPLOYEE records are sorted on the join attribute (DEPT#) before the join operation occurs. Then the participating PROJECT records with a given department number need only be retrieved one time.

Since the parameter JVAL controls the "degree" of the many-to-many relationship between EMPLOYEE and PROJECT, we expect to see the greatest difference between the two methods occurring when JVAL is high. Graph 4 confirms this by plotting page accesses against file size for three values of JVAL.

The added sorting cost for Case 3 is highest when the restriction on EMPLOYEE has little or no effect. Graph 5 makes the same comparisons as the previous graph, changing only the value of RSEL to 1.0. Even with high sorting costs, the use of cases 2 and 3 appears highly advantageous.

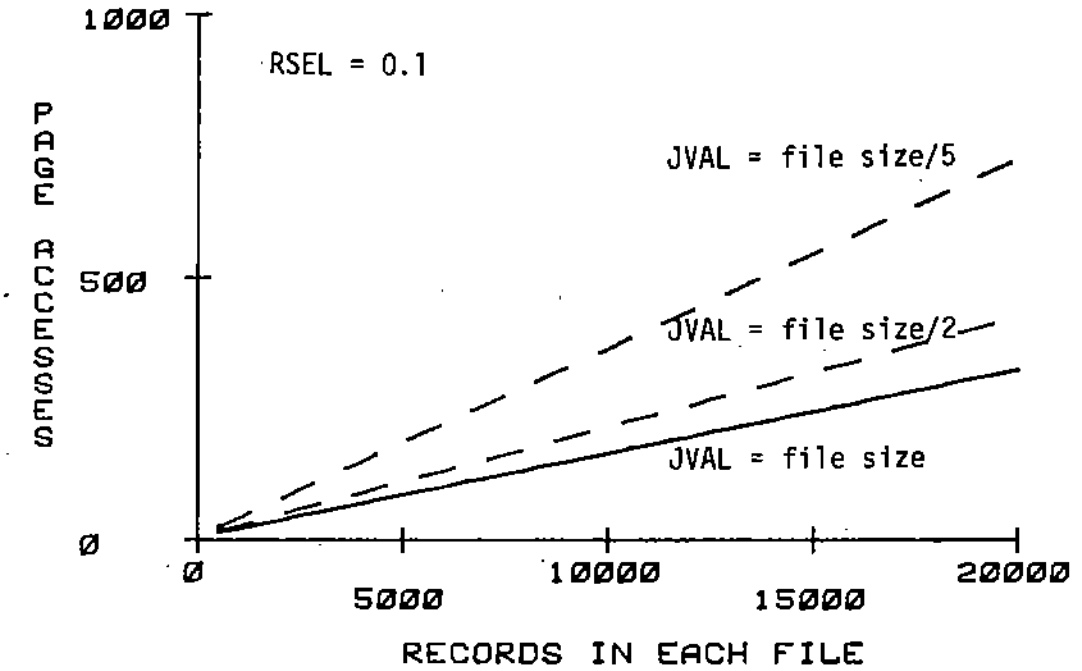


Figure 12. Cases 3 and 2 (solid) and Astrahan (dashed).

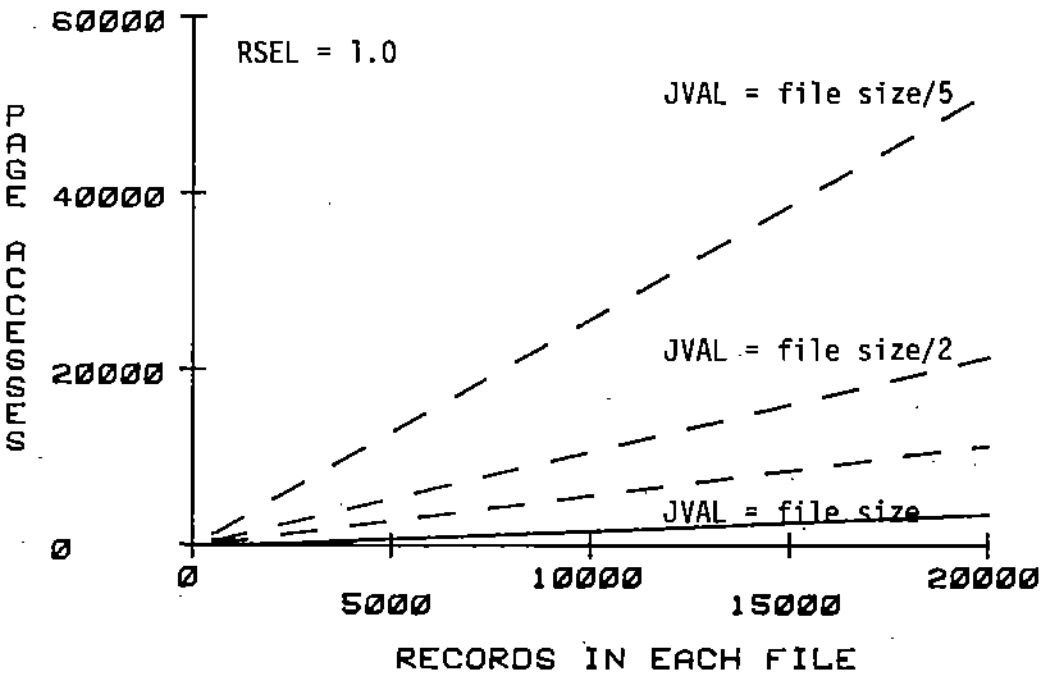


Figure 13. Cases 3 and 2 (solid) and Astrahan (dashed).

In addition to varying these two parameters, numerous changes in the index structures were also examined (e.g. no restriction indexes, non-clustering join indexes). The results all pointed to the same conlusion: even though the Astrahan access path is similar to that offered by our cases 3 and 2, the redundancy in its join operation results in a costlier access path when dealing with many-to-many relationships.

Smith and Chang.

Smith and Chang [12] have addressed the problem of optimizing access paths in a relation a database system. This optimization is accomplished for a given query through determining both in what order the relational operations should be performed, and at what places in the access path a sort would be beneficial. Because their procedure is well defined by a set of rules, we can again take our general relational query, construct an access path using their scheme, and compare its performance with similar paths from our model under identical cost assumptions.

In developing rules for path optimization Smith and Chang have adhered to some basic principles which include 1) using indexes on files whenever possible to avoid unnecessary record retrievals; 2) reducing the size of the files being accessed as soon as possible via restrictions and projections, in order to decrease the amount of temporary

storage needed during the access process; and 3) sorting the intermediate results whenever such a sort would improve the efficiency of the following operation. By following these principles and the resulting rules we arrive at an "optimal" access path for the general query consisting of two restrictions, a join and a projecton. Since Smith and Chang have modularized the access operations in a way similar to ours, we can state their solution using our notation:

file 1 RI—DR—Pj—S—JF—X—P

file 2 RI—DR—Pj—S—JF—X—P

This is identical to our path 3-B. If we had not followed thier recommendation to perform a partial projecton early in the path, we would have our Case 3-A:

RI—DR—S—JF—X—P

RI—DR—S—JF—X—P

If we had not followed their recommendation to restrict the files as soon as possible, we would have our Case 4-C:

Scan S——JF—— X——RF——P

Scan S—— JF—— X —— RF—— P

In comparing these three access paths using the cost equations presented earlier, we have verified that path 3-B does, in fact, require fewer secondary storage accesses. Figure 14 illustrates these results, showing the costs of the three paths as a function of file size. The situation shown here has non-clustering restriction indexes. The restriction selectivity is varied from 0.1 to 0.5 to 1.0 in the graph. Where the restriction is very effective, the cost differences contrast primarily the number of record retrievals required via restriction indexes versus via segment scans. Where RSEL is 1.0 all records are retrieved anyway, and the cost differences reflect the varying sorting costs incurred by the paths.

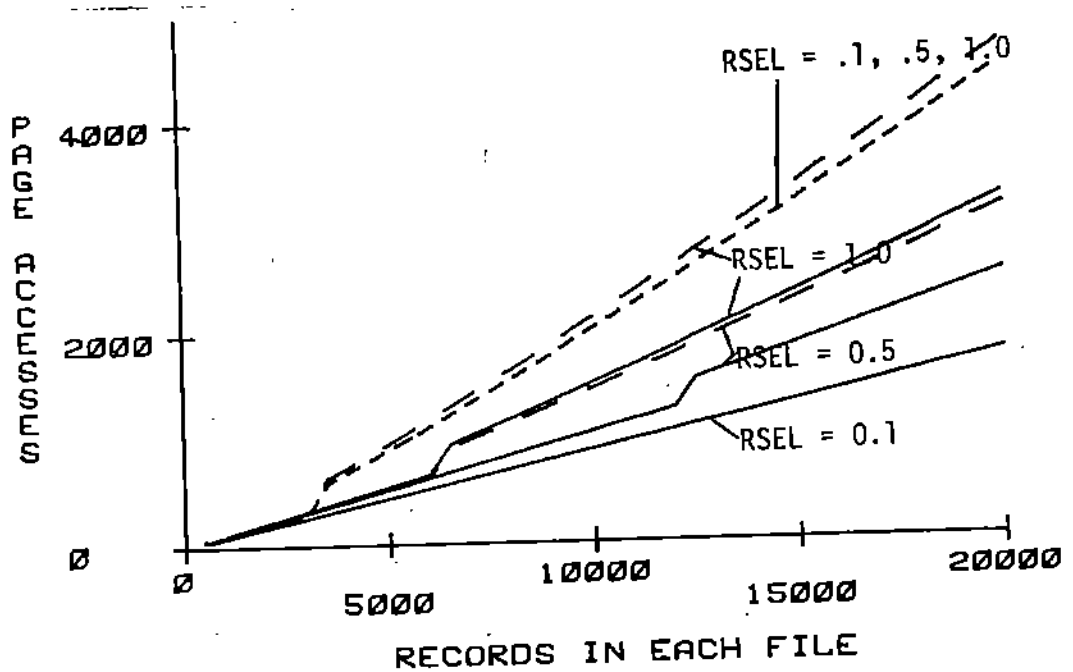


Figure 14. Smith and Chang: Cases 3-B (solid), 3-A (long dashed), and 4-C (short dashed).

There is an important cost parameter, however, which Smith and Chang fail to consider in their optimizing rules. They suggest that restrictions and projections should appear early in the access path in order to reduce the size of the temporary files being used, but they neglect the fact that the join operation can also eliminate records from consideration. Recall, for example, our previous query with employees and projects. If only certain departments had projects to work on, many employee records could be eliminated by checking first to see which employee records would participate in the join with the PROJECT file. We provide for this option, for example, in our Case 1, where

the first step filters out those records which do not participate in the join:

file 1 JI——DR——RF——Pj——X——P

file 2 JI——DR——RF——Pj——X——P

If the join operation is more selective than the restrictions and projectons, it will be clearly advantageous to perform the join filter ahead of restrictions and projections. In the next section we do, in fact, show examples where our Case 1 requires fewer secondary storage accesses than Case 3-B (eg. Figure 20).

B. SAMPLE COMPARISONS OF ACCESS PATH COSTS.

Our attention now focuses on the model itself, and how it can be used for the design and selection of access paths in a database system. If applied to an existing system, the parameters describing the database environment will take on fixed values, and, in fact, the number of access paths available may be limited by such factors as the organization of the database. In this situation the model can be used to select an optimal access path from those available, for various query characteristics. For example, one path may be

particularly efficient when the restriction selectivity of a query is very low, while another may excel with higher selectivity values. The obvious procedure is to assign values to the known parameters, and then use the cost equations to compare the available access paths' performances under various query types. The model can also be used in a much more comprehensive since in the design phase of a database system. The wide range of parameters allows the designer to investigate relationships between path performance and storage organization, as well as query characteristics.

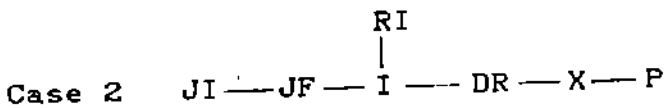
For purposes of illustration we make three sample comparisons of various access paths. In doing so we highlight several parameters which affect the relative costs of the paths. First we carefully examine the performance of paths from Cases 1, 2, and 3 of the model. In the second example we investigate the performance of some options available in Case 4. Finally we compare Cases 5 and 6, where linking structures exist among the files. We believe that the parameters chosen for these examples are "typical," but obviously the values would change for various database applications.

CASES 1, 2, AND 3.

These three cases all utilize indexes during the record retrieval process:

Case 1 JI — JF — DR — RF — X — P

Case 2



Case 3 RI — DR — Pj — SR — JF — X — P

For Case 1 we assume that there is no mechanism to provide for feedback, an option described earlier. In Case 3 we have chosen path B, where a partial projection is performed ahead of the sort. With the cost assumptions we have made, this path will always be less expensive than that which saves the entire projection for the last step.

For the sake of comparison, each of these access paths need only be applied to a single file. Two files, of course, are actually needed for the join operation, but the cost of each of these paths applied to one file is quite independent of the access path chosen for the second file.

In analyzing the performance of these paths we have chosen to vary four of the model's input parameters:

(1) join selectivity

(2) restriction selectivity

(3) clustering and non-clustering indexes

(4) file size.

a) Constant Size; Various Clustering Indexes

In the first group of tests we hold the file size at a fixed 2000 records. Then for each combination of clustering indexes we vary the restriction and join selectivities.

(1) Non-clustering Join Index, Non-clustering Restriction Index. Figure 15 shows the number of page accesses vs join selectivity, where the restriction selectivity is set at 0.1, and neither the join index nor the restriction index is clustering. Case 3-B incurrs all of its costs before the join operation occurs, so it is not affected by a change in the join selectivity. Case 1, on the other hand, uses a join index in the record retrieval process. When only a few records participate in the join, then, only a few pages need to be retrieved from storage. However, since the index is non-clustering a small increase in the percentage of join participants can mean a dramatic increase in the number of pages accessed. Thus by the time that the join selectivity reaches 25%, all of the file's pages are being accessed, and Case 1 has reached its maximum cost. Since Case 1 is not affected by the relatively small restriction selectivity, its maximum cost exceeds that of Case 3-B. Case 2 also uses a join index, but its costs are kept low through the additional use of a restriction index. All filtering is done before any retrievals, so the minimal number of records are retrieved. Notice that the cost of Case 2

does exceed that of path 3-B, however, when the join selectivity is near 100%. The reason is that at this point the join index and filter are of little use to path 2, since all of the records are being retrieved. Thus the two paths become very similar; except that Case 2 must still retrieve and use the join index. So Case 2 has an additional index retrieval cost, while Case 3-B's extra sort cost is marginal, due to the file size.

In Figures 16 and 17 the situation is the same, except that the restriction selectivities are set at 0.5 and 1.0, respectively. Case 1 does not change because all of its costs are incurred before the restriction filter takes place. Path 3-B becomes more expensive simply because more records pass the restriction and must be retreived. The cost of Case 2 rises more rapidly since the restriction index is less effective at filtering out additional records. When both restriction and join selectivities are near 100%, the extra indexes of Case 2 give no advantage over the other cases, and, in fact, only add extra index retrieval costs.

It is interesting to note that already in these first three graphs we have situations in which each of the three paths can out perform both of the others.

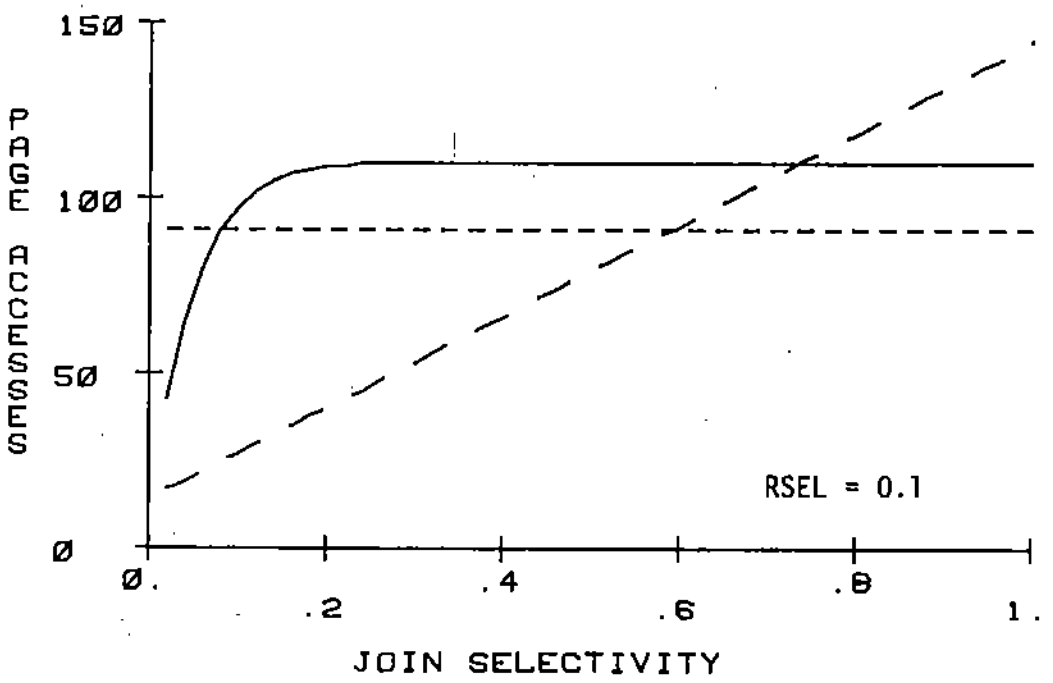


Figure 15. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

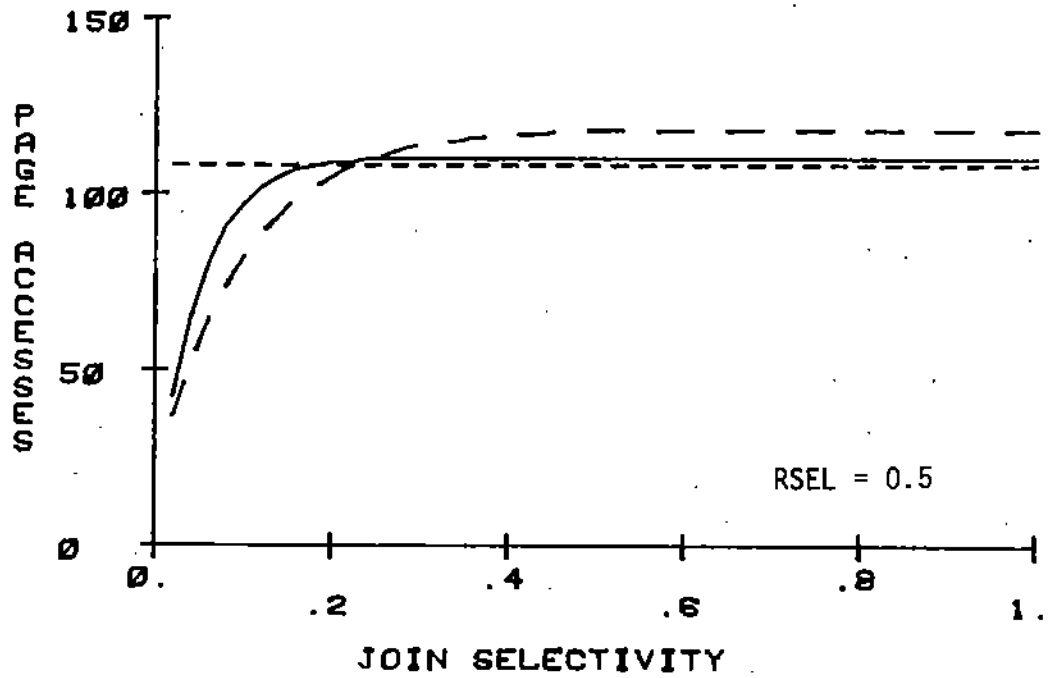


Figure 16. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

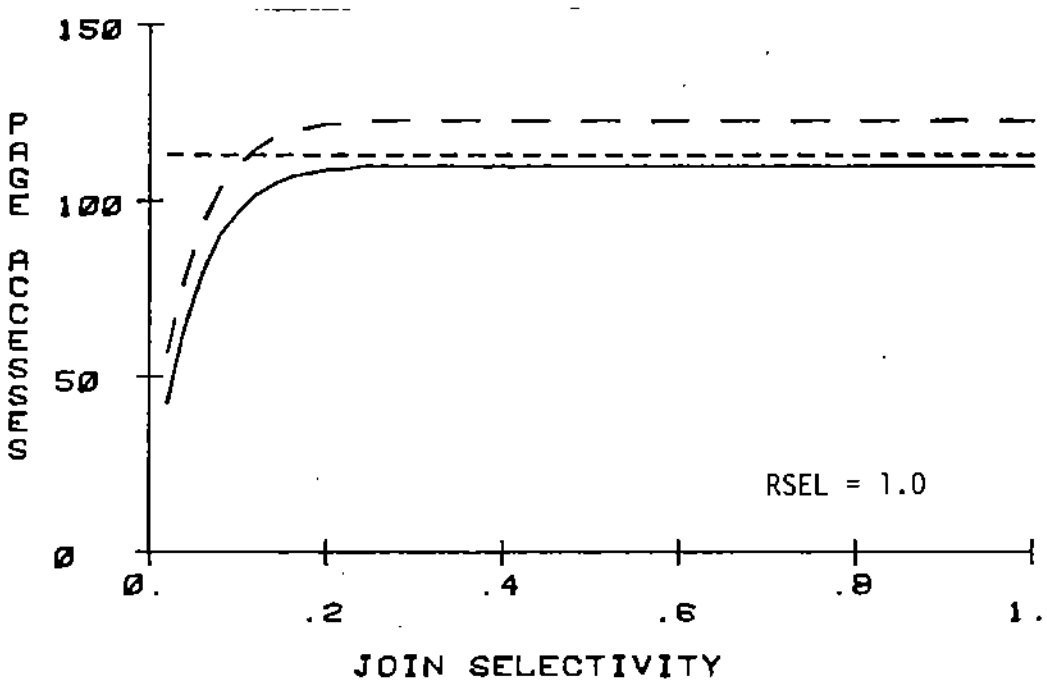


Figure 17. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

(2) Clustering Join Index, Non-clustering Restriction Index.

We now repeat the above comparisons, changing only the parameter for the join index from a non-clustering state to a clustering state. The restriction selectivity is set to 0.1 in Figure 18, 0.6 in Figure 19, and 1.0 in Figure 20. Case 3-B again does not change with the join selectivity since it does not utilize a join index. Paths 1 and 2 now have linear costs. Because the join index is clustering, the number of page accesses needed to retrieve the data is directly proportional to the

percentage of the file passing the join filter. The additional restriction index which Case 2 retreives and uses is of value only in Figure 18, where the restriction selectivity is 0.1. Because the records to be retrieved are clustered together in pages, the restriction filter does not otherwise significantly reduce the number of pages that must be retrieved.

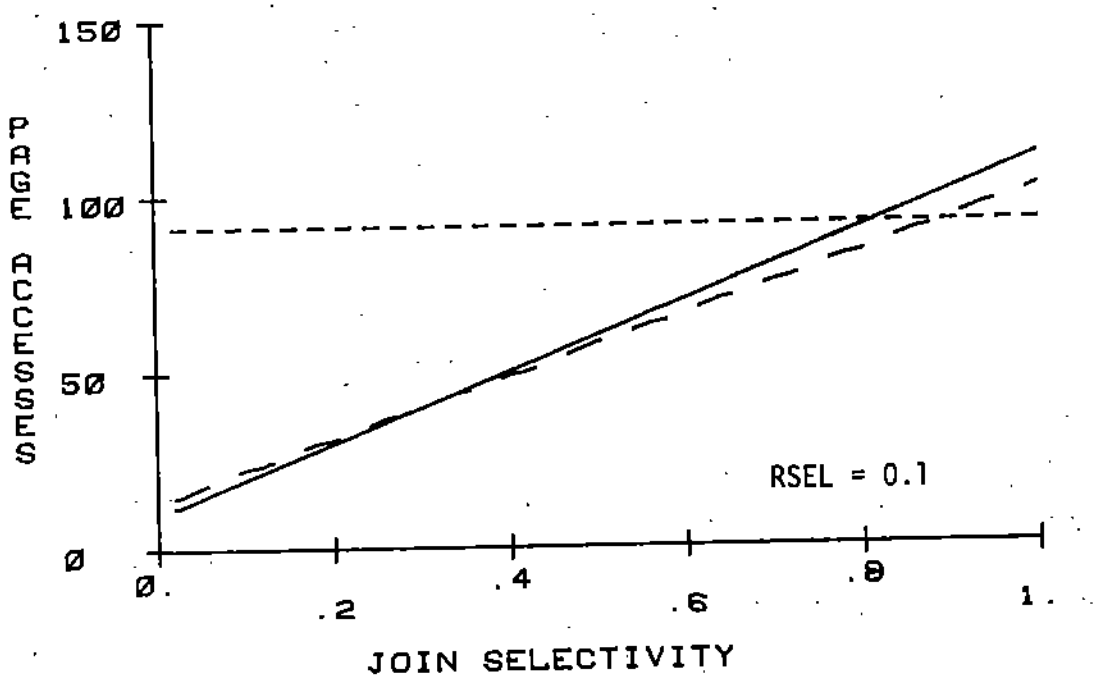


Figure 18. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

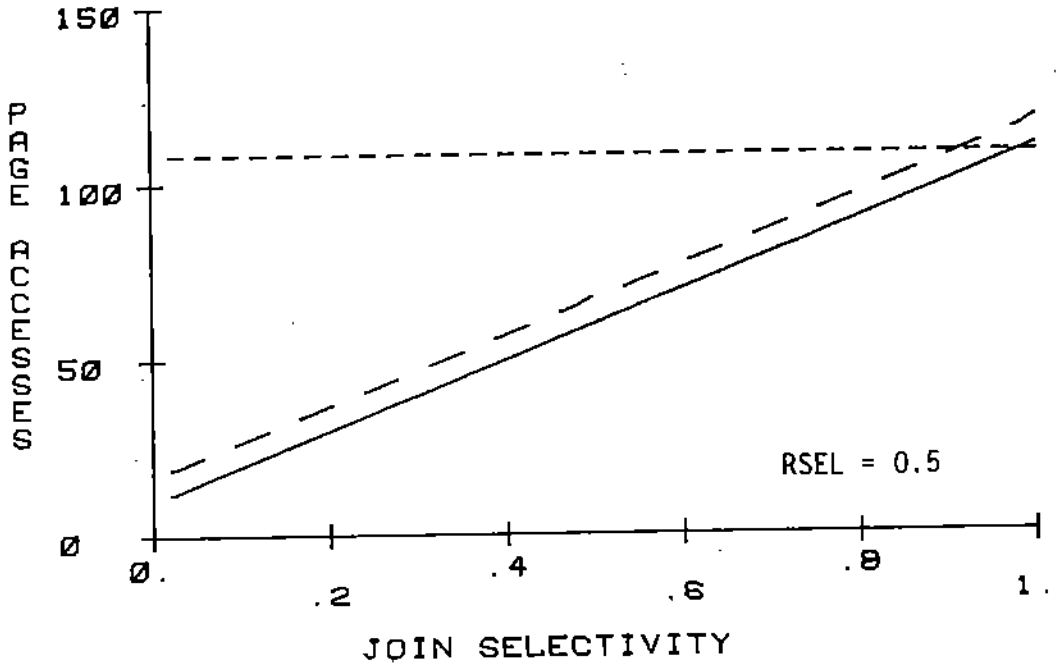


Figure 19. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

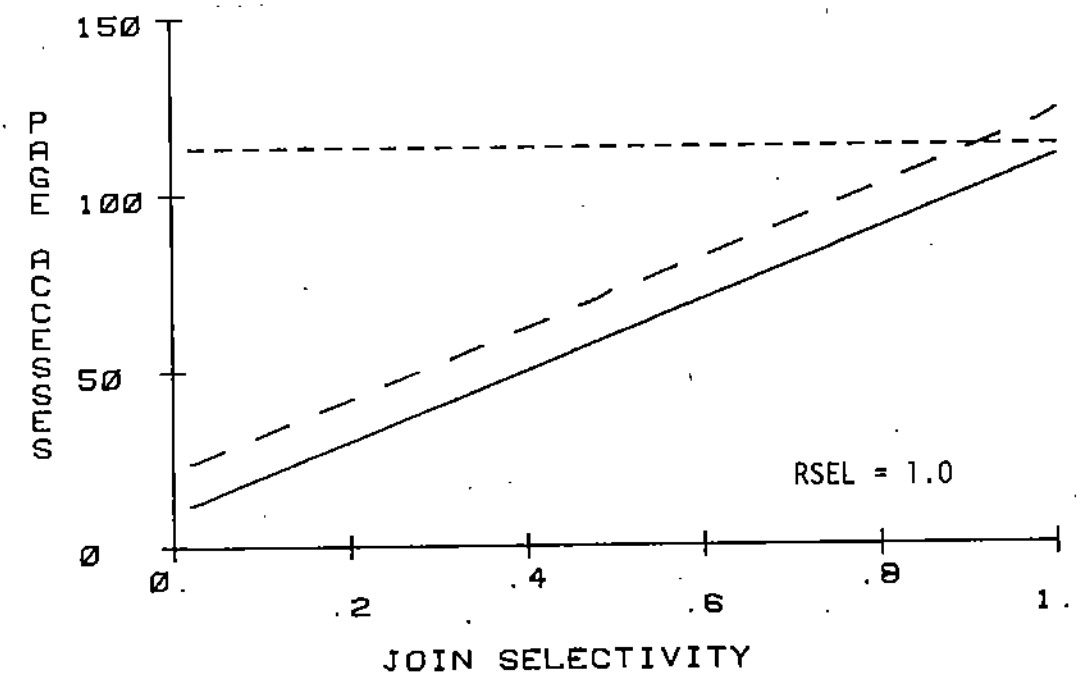


Figure 20. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

(3) Non-clustering Join Index, Clustering Restriction Index.

The same comparisons are again repeated in Figures 21-23, but this time with a clustering restriction index, and a non-clustering join index. Since path 1 does not use a join index, its performance repeats that of Figures 15-17, where there was no clustering. Case 3-B, which fully utilizes the restriction index, is greatly improved by the clustering when the restriction eliminates part of the file. The additional join index which Case 2 uses is beneficial only for extremely small join selectivity values. Only then will the index reduce the number of pages which must be accessed. When both restriction and join selectivites have values near 100%, each of the cases must retrieve all of the pages of the file. The indexes must be retrieved in addition, and since Case 2 uses both indexes, it has the highest cost.

In summary, a clustering restricting index gives an advantage to Case 3-B, while path 1 is the least expensive when the join index is clustering. Case 2 is a close second in both instances. When no clustering exists, the restriction and join selectivity values determine which path should be used. can out perform both of the others.

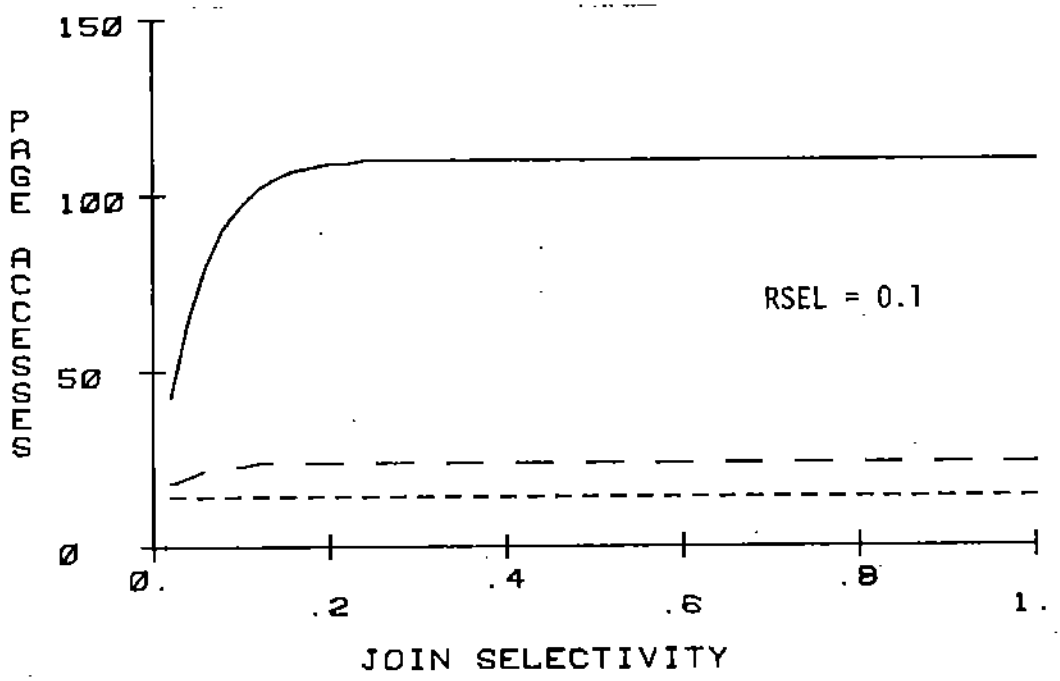


Figure 21. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

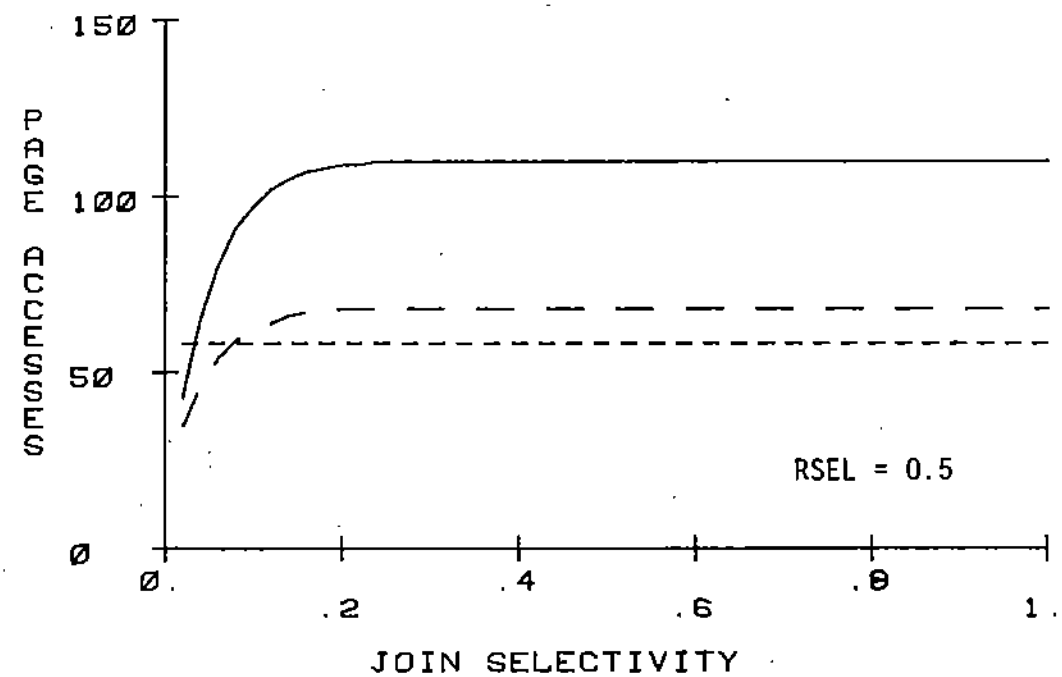


Figure 22. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

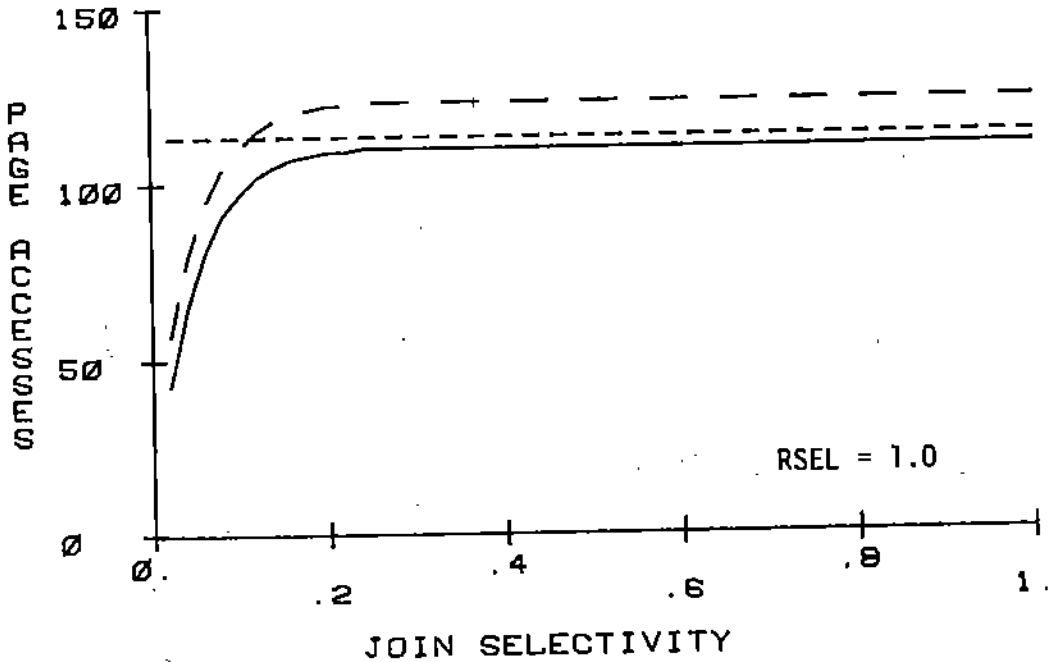


Figure 23. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

b) Various sizes; Constant Non-clustering Indexes. In this second set of tests comparing Cases 1,2 and 3-B we investigate the effect of file size on the relative path costs. Non-clustering indexes are maintained, and now file size, ranging from 500 to 20,000 records, is plotted against the number of page accesses needed to complete the access paths.

In Figure 24 the restriction selectivity is a constant 0.1, and the join selectivity is also set to 0.1. The figure shows that the relative costs of the three access paths are maintained over a wide range of file sizes. Variations in the join and restriction selectivities seem generally to verify this result.

Setting both selectivity values to 0.5, however, yields a more interesting finding. As Figure 25 shows, the access costs are very similar for the three cases. However, the relative cosets actually change several times, the most visible change occuring at a file size of 12,000. At this point path 3-B switches from an internal sort to an external sort of the retrieved records. These sorting costs are not linearly, but logarithmically proportional to file size. Thus when path costs are relatively close, this added sort cost becomes an important factor with large file sizes.

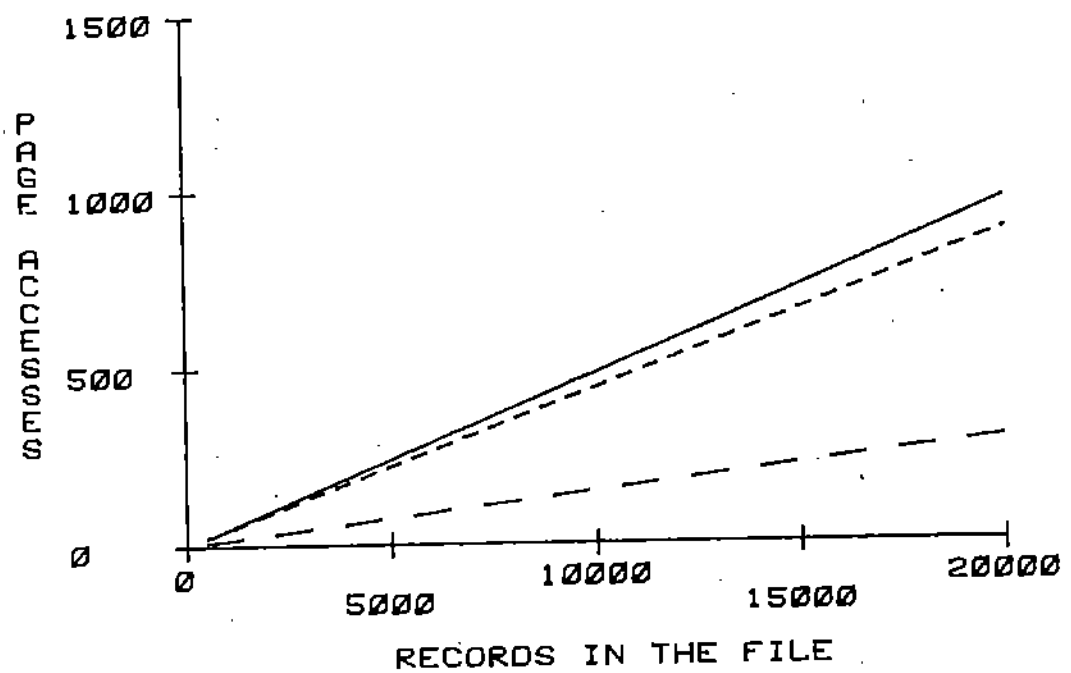


Figure 24. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

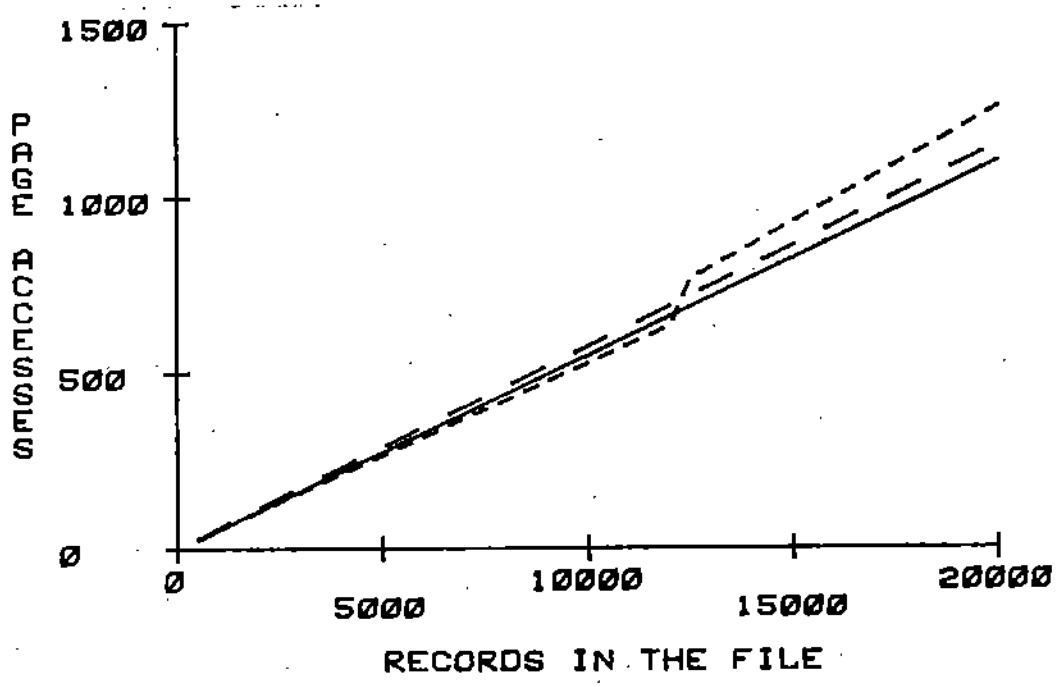
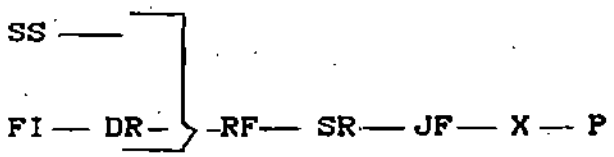


Figure 26. Cases 1 (solid), 2 (long dashed), and 3 (short dashed).

CASE 4.

We now move on to Case 4 and make a more brief comparison of its alternatives. First, we recall that there are several retrieval methods that can be used for this case. We will examine the clustering and non-clustering full index retrieval, and the segment scan under several different conditions for Case 4-A:



As in the previous comparisons, we need only apply the access path to one file, since we are only interested in relative costs. In Figures 26 and 27 the number of page accesses is plotted against the size of the file, which again ranges from 500 to 20,000 records. The parameter which distinguishes the graphs is the amount to which the file is clustered with other files in the database. Recall that file clustering means that records from several files are physically interleaved (eg. in hierarchical order) on the pages of the database.

The situation in Figure 26 involves a file which is not clustered with any other files. Each retrieval method accesses exactly all of those pages containing the file. However, the methods involving clustering and non-clustering full indexes must retrieve the index pages as well. Thus their total cost is higher.

In Figure 27 the file is clustered at a ratio of 1 to 50 with other files. Since only 20 records are allowed to fit on a page, the cluster contains many pages which do not have any records from the file to be retrieved. The segment scan searches the whole cluster, and thus makes many more page accesses than is necessary. The clustering index retrieval performs the best, since it retrieves only those pages containing the desired records, and since it retrieves those records in their physical order. We see from this test case that the amount of file clustering is another important

parameter in access path cost.

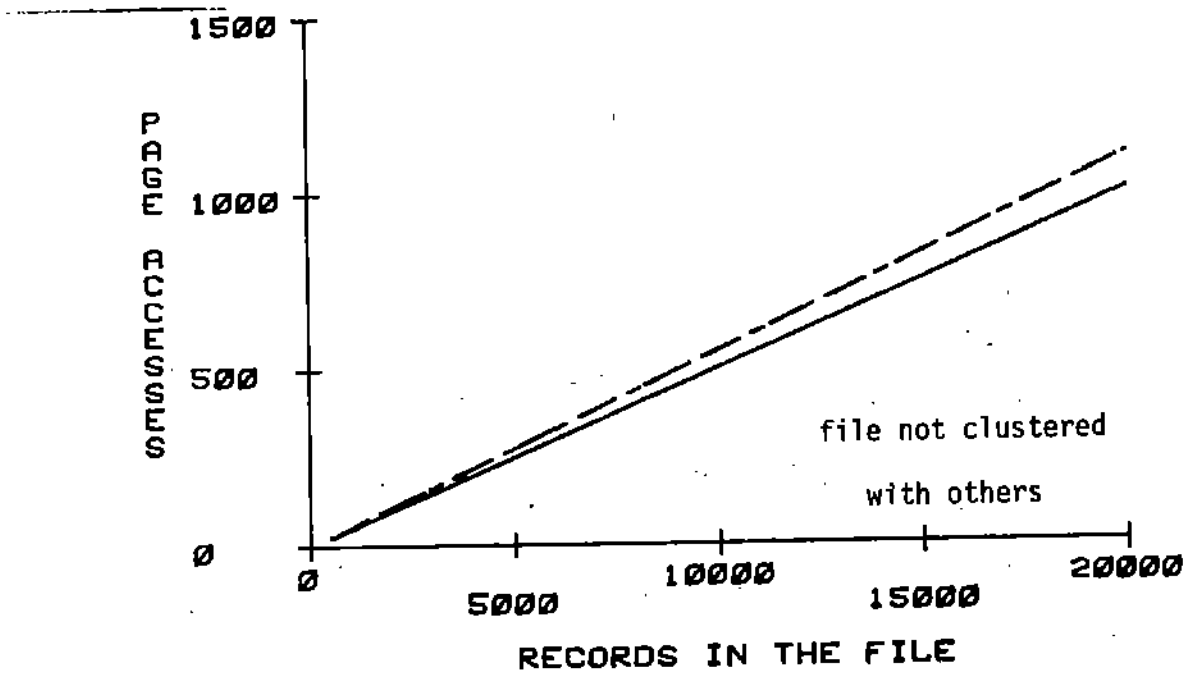


Figure 28. Case 4-A using segment scan (solid), clustering (dashed), and non-clustering (dashed) indexes.

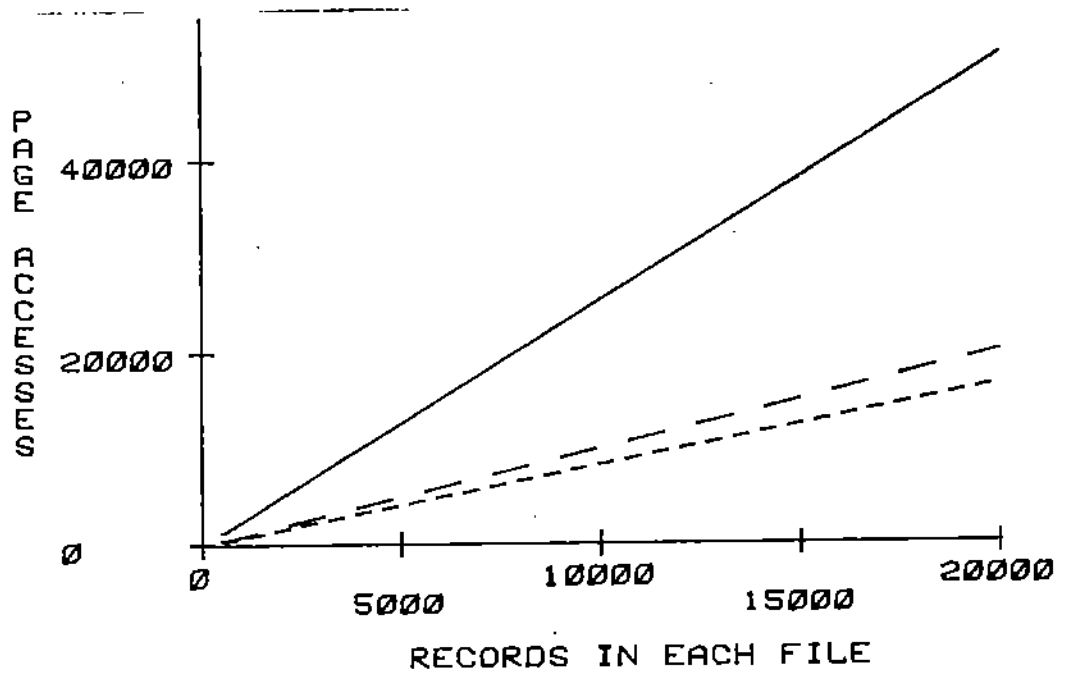


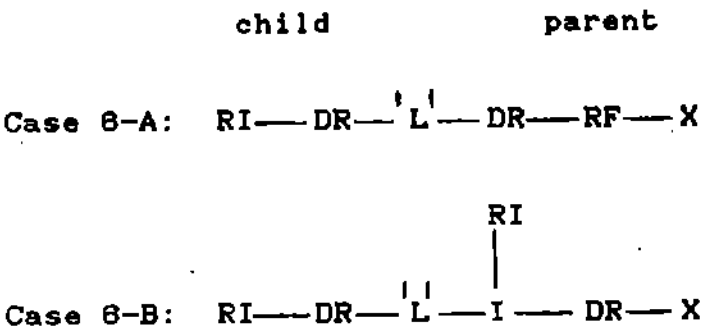
Figure 27. Case 4-A using segment scan (solid), clustering (dashed), and non-clustering (dashed) indexes.

CASES 5 AND 6.

For the final example of path comparisons we look at Cases 5 and 6, which take advantage of logical parent-child relationships between two files. In order to limit the scope of this comparison we choose one of the possible methods from Case 5, and two from Case 6:

parent child

Case 5: RI— DR—$^{1}L^{M}$— DR— RF —X



Thus in all cases we assume that a restriction index exists on the first file, and that it is, in fact, a clustering index. The parameters whose affect we investigate are the following:

(1) clustered or non-clustered files

(2) clustering or non-clustering twin links

(3) restriction selectivity

(4) parent-child fanout.

a) Non-clustered Files

We begin with parent and child files which are not clustered. The twin links in the child file are also not clustered, meaning that the child file is not sorted on the join, or link, attribute. The size of the parent file is set at 2,000 records, and the restriction selectivity is varied from .02 to 1.0 This parameter is plotted against the number of secondary storage accesses.

Figure 23 shows the results of a parent-child fanout of 1 to 1. Since there is only one child per parent, the two cases 5 and 6-A become identical. Case 6-B has the

advantage of restricting the child file before record retrieval is performed on the child. Thus fewer pages need to be accessed. As the restriction becomes less effective, however, this cost advantage diminishes.

In Figure 29 a parent-child fanout of 1 to 100 has been implemented. All other parameters are the same. The child file, which is now 100 times larger than the parent file, has a significantly larger restriction index than that of the parent file. The added page accesses in retrieving this index, along with the larger child file account for the higher path cost of Case 6-A.

Keeping these parameters constant, we now suppose that the twin pointers in the child file are clustering pointers, so that all of the twins are physically adjacent. Figure 30 shows that the change has significantly lowered the cost of path 5 while leaving Cases 6-A and 6-B only slightly better off. This is due to the fact that only Case 5 makes use of twin pointers in the child file. As one would expect, though, when the fanout is again reduced to 1 to 1, Case 5 no longer can use twin links, and its advantage is lost.

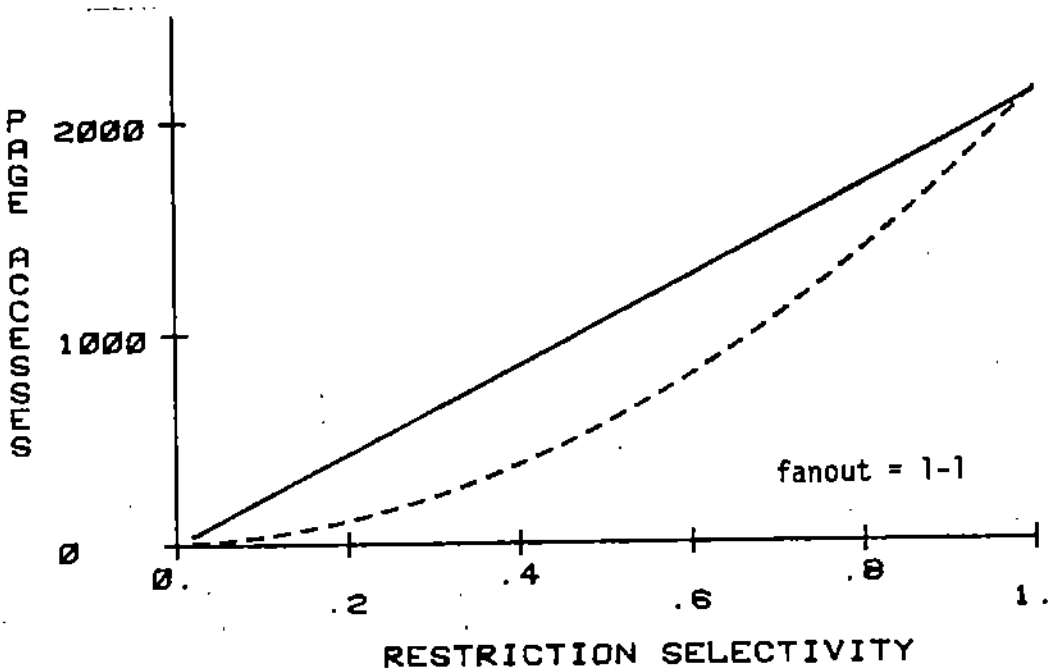


Figure 28. Cases 5 (solid), 6-A (solid), and 6-B (short dashed).

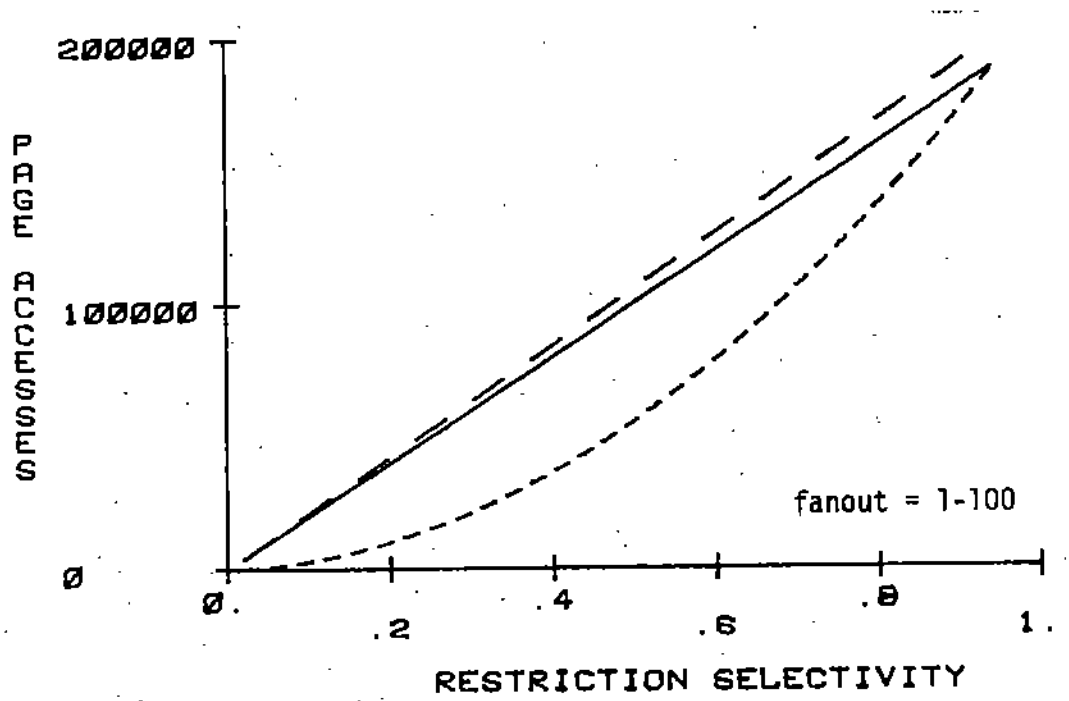


Figure 29. Cases 5 (solid), 6-A (long dashed), and 6-B (short dashed).

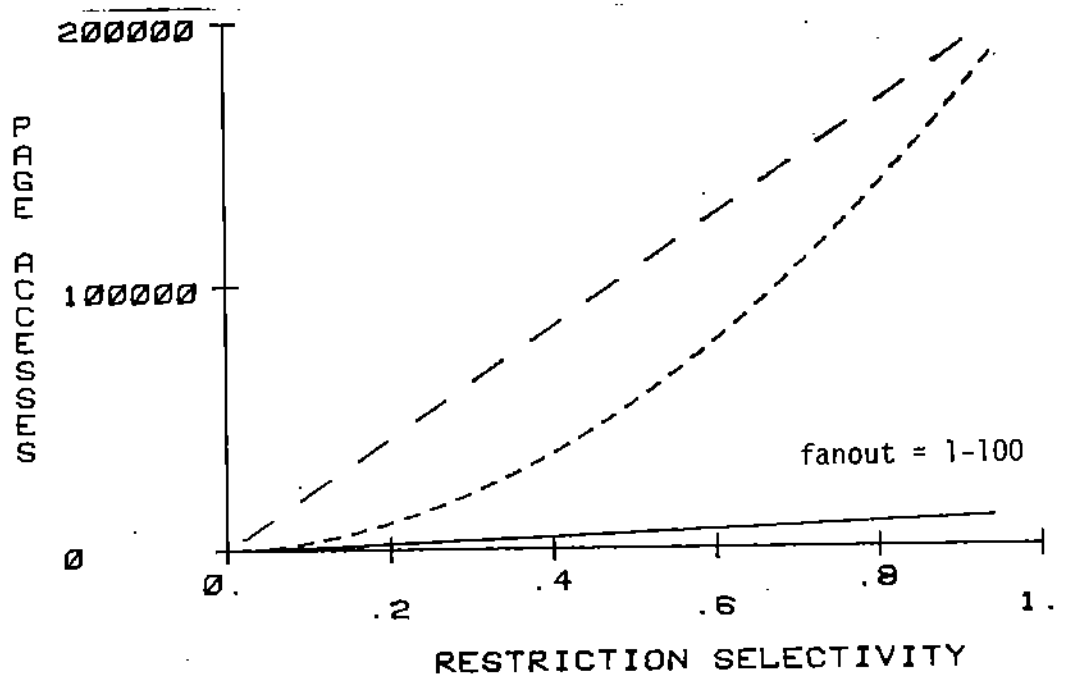


Figure 30. Cases 5 (solid), 6-A (long dashed), and 6-B (short dashed).

b) Clustered Files.

We now examine the situation in which the parent and child files are clustered. Recall that this implies clustering twin pointers in the child file, so long as there are only two files in the cluster. The restriction selectivity is again varied from 0.02 to 1.0, and the parent file contains 2,000 records.

The situation for Figure 31 has a fanout of 1 to 1, which again makes Cases 5 and 6-A equivalent. The access costs are dramatically reduced from the previous trials because a new page seldom needs to be accessed in tracing the link

between parent and child. For case 6-B, the restriction index on the parent file is of little benefit. In fact, the cost of retrieving the index makes the cost of the case greater than that of the other two.

When the fanout factor is again raised to 1 to 100, Case 5 is again most economical. As explained above for Figure 30, this cost advantage is due to the fact that the twin records are physically clustered. Only a few page accesses are needed for a parent record to link to all of its children. The Case 6 paths, however, only allow a single child at a time to be retrieved and then linked to its parent.

Case 5, then, appears to perform better than the Case 6 paths whenever twin clustering is present in the child file. Case 6-B , on the other hand, puts its extra restriction index to a good advantage when twin clustering is not present. These results have been further confirmed by similar tests with larger files.

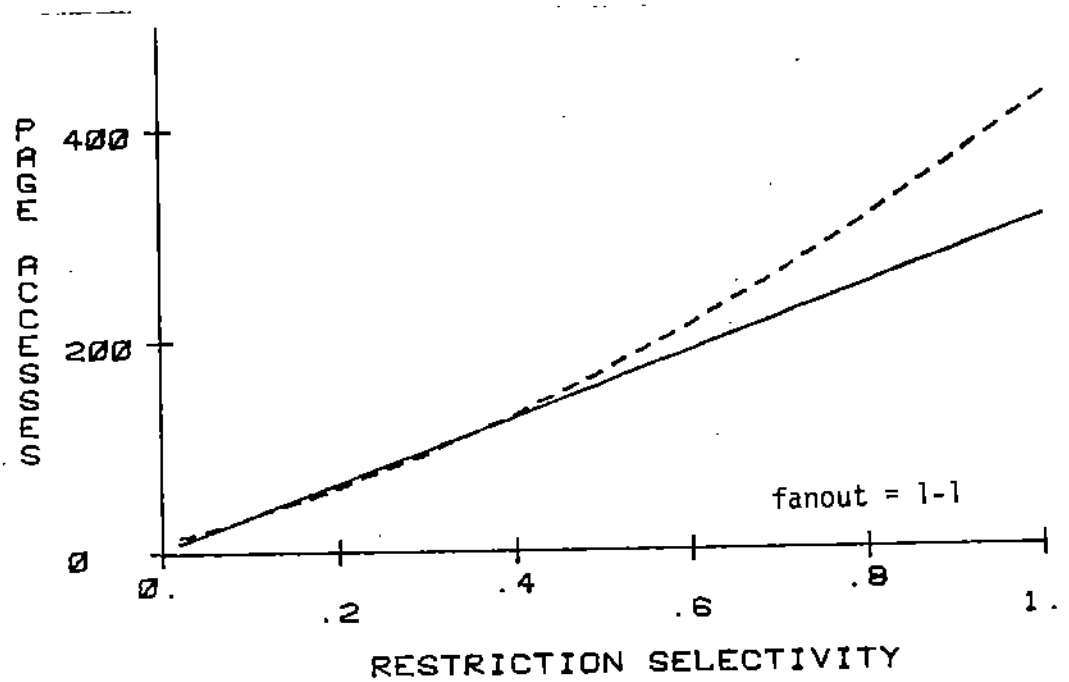


Figure 31. Cases 5 (solid), 6-A (long dashed), and 6-B (short dashed).

Table 1. Constant parameter values

PARAMETER	VALUE
MEM	25
BL	6
Z	3
PR	100
NLEV	3
PI	200
RVAL	1/RSEL
Pj	0.5
Prj	0.6

Table 2. Varied parameter values.

FIGURE	9	10	11	12	13	14	15	16
S(file 1)	v	v	v	v	v	v	2000	2000
S(file 2)	v	v	v	v	v	v		
files in SCLUS(1)	1	1	1	1	1	1	1	1
files in SCLUS(2)	1	1	1	1	1	1		
JVAL	S(2)	S(2)	v	v	v	S1/10		
RSEL	1,.1	1,.1	.1	.1	1	v	.1	.5
JSEL	1	1	1	1	1	1	v	v
Rest Inx(1)				c	c	n	n	n
Join Inx(1)		c	c				n	n
Other Inx(1)	c							
Rest Inx(2)						n		
Join Inx(2)		c	c	c	c			
Other Inx(2)	c							
Files 1&2 Clustrd	no	no	no	no	no	no		

v = varied
c = clustering
n = non-clustering

Table 2, cont.

FIGURE	17	18	19	20	21	22	23	24
S(file 1)	2000	2000	2000	2000	2000	2000	2000	v
files in SCLUS(1)	1	1	1	1	1	1	1	1
RSEL	1	.1	.5	1.	.1	.5	1.	.1
JSEL	v	v	v	v	v	v	v	.1
Rest Inx(1)	n	n	n	n	c	c	c	n
Join Inx(2)	n	c	c	c	n	n	n	n

FIGURE	25	26	27	28	29	30	31
S(file 1)	v	v	v	2000	2000	2000	2000
S(file 2)				2000	200K	200K	2000
files in SCLUS(1)	1	1	many	1	1	1	2
files in SCLUS(2)				1	1	1	2
FAN				1	100	100	1
RSEL	.5			v	v	v	v
JSEL	.1			1	1	1	1
Rest Inx(1)				c	c	c	c
Other Inx(1)	v	v					
Rest Inx(2)				c	c	c	c
Files Clustrd		no	yes	no	no	no	yes
Child-twin clust				no	no	yes	yes

v = varied
c = clustering
n = non-clustering

VI. CONCLUSIONS

We believe that the access path model described in this paper is successful in generalizing the access operations which are performed on a relational database. The modularization of these operations allows them to be subsequently linked together into access paths in a countless number of ways. This leaves great flexibility for comparing old and even for designing new access paths.

The parameters which we have described allow the user to specify important characteristics of the queries into the database, as well as the physical organization of the database. While it is true that many of these parameters consist of average values, we feel that this is sufficient to provide an accurate analysis while keeping the model relatively simple. The accompanying cost equations reflect our view that the secondary storage access is still the overriding factor in the performance of an access path. Since these equations are also in modular form, modifications to suit particular retrieval capabilities are easily made.

The implementation and testing of the model has shown us several things. First, in comparing the access paths with

others previously described we have found many similarities in both design and cost evaluation. On the other hand, we have also been able to demonstrate that several parameters which are unique to this model have decided effects on the path performances. Thus we are better able to predict the access path costs. By comparing various access paths of the model under numerous circumstances, we have seen that no single path is "best" under all circumstances. Only slight variations in the database environment can alter the paths' relative costs. For this reason a cost evaluation model such as this can be of great assistance in the design and selection of database access paths.

If the DBMS is capable of supporting multiple storage organiations and access operations, a major goal would be to include a cost evaluation package such as this for real time use. So long as the computation involved remains fairly simple, the choosing of an optimal path for each database query could significantly reduce the retrieval cost.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Astrahan, M. and D. Chamberlin, "Implementation of a structured English query language," Comm. ACM, Vol. 18, No. 10 (Oct. 1975), pp. 580-588.

2. Astrahan, M. et al., "System R: A relational approach to data base management," ACM Transactions on Database Systems, Vol. 1, No. 2 (June 1976), pp. 97-137.

3. Blasgen, M. and K. Eswaran, "Storage access in relational data bases," IMB Systems J., No. 4, 1977, pp. 363-377.

4. Cardenas, A., "Evaluation and Selection of File Organization--A Model and System", Comm. ACM, Vol. 16, No. 9 (Sept. 1973), pp. 540-548.

5. Codd, E., "A relational model of data for large shared data banks," Comm. ACM, Vol. 13, No. 6 (June 1970), pp. 377-387.

6. Date, C., An Introduction to Database Systems, Addison-Wesley, 1976.

7. Hsiao, D. and Harary, F., "A formal System for Informaton Retrieval from Files", Comm. ACM, Vol. 13, No. 2 (Feb. 1970), pp. 67-73.

8. Precherer, R., "Efficient evaluation of expressions in a relational algebra," Proc. ACM Pacific 75 Conf., April 1975, pp. 44-49.

9. Rothnie, J., "Evaluating inter-entry retrieval expressions in a relational data base management system," Proc. AFIPS 1975 NCC, Vol. 44, Montvale, New Jersey: AFIPS Press, pp. 417-423.

10. Schkolnick, M., "A clustering algorithm for hierarchical structures," ACM Transactions on Database Systems, Vol. 2, No. 1 (March 1977), pp. 27-44.

11. Severance, D., Some Generalized Modeling Structures for Use in Design of File Organizations, Ph.D. Dissertation,

APPENDICIES

The University of Michigan, 1972.

12. Smith, J. and P. Chang, "Optimizing the performance of a relational algebra database interface," Comm. ACM, Vol. 18, No. 10 (October 1975), pp. 568-579.

13. Yao, S., "An attribute based model for database access cost analysis," ACM Transactions on Database Systems, Vol. 2, No. 1 (March 1977), pp. 45-67.

14. Yao, S., "Approximating block accesses in data base organizations," Comm. ACM, Vol. 20, No. 4 (April 1977), pp. 260-261.

15. Yao, S., "Optimization of Query Evaluation Algorithms", in preparation, 1978.

APPENDIX A.

The parameter PGS(I,J) has been defined as the average fraction of a page traversed in scanning from a record , r_i, in file I to the next record, r_j, in file J. For example, if PGS(K,K) = 0.2, then it will take, on the average, 1 page access to scan from a record of file K to the next five consecutive records of K.

If the two files are not physically clustered, then there is always exactly 1 page access required to reach the desired record, r_j. If the files are clustered, in hierarchical fashion, then records from several files may reside in the same memory page.

Yao has used the following parameters to calculate the distance travelled in scanning in hierarchical sequence from r_i to r_j:

t_i	record size for file I in some units, u
p	number of units, u, per page of memory
x_{ij}	1 if I and J are physically clustered 0 otherwise
f_{ij}	average hierarchical fanout between I and J

S_i set of all hierarchical "child" files of I.

The following recurrence relation gives us the distance in units, u, between two records of the same file:

$$D(I,I) = t_i + \sum_{j \in S_i} f_{ij} * D(J,J) * x_{ij}.$$

Then D(I,I)/p is that same distance in pages. Since no more then 1 page access is ever required in getting a single record, we have:

$$PGS(I,I) = \min\{1, D(I,I)/P\}.$$

For PGS(I,J) we need to traverse through the child records of I which are <u>between</u> r_i and r_j. In this situation the distance is expressed as

$$D(I,J) = \begin{cases} 1 & \text{if I and J are not clustered} \\ t_i + \sum_{\substack{K \in S_i \\ K < J}} f_{ik} * D(K,K) * X_{ik} & \text{otherwise.} \end{cases}$$

where k<j means that K appears before J in hierarchical sequence. Then

$$P(I,J) = \min\{1, (D(I,J)/p) * x_{ij} + (1-x_{ij})\}.$$

APPENDIX B.

We wish to determine the number of page accesses required in retrieving a set of K records which are randomly spread among M pages containing a total of N records. We assume that we are given a list of the entire set of K records at the outset, so that a particular page containing several of the desired records need only be retrieved once.

Cardenas has suggested the expression

$$P''(N,M,K) = M(1-(1-1/M)^K).$$

Yao[14] summarized several results which show this expression to be unsatisfactory, and has derived the exact solution as:

$$P(N,M,K) = M[1-\prod_{i=1}^{K}(N*d-i+1)/(N-i+1)]$$

where $d = 1 - 1/M$.

The evaluation of this functon, however, is quite costly in terms of computing time, and so a simpler approximation is desired. We have used the function

$$P'(N,M,K) = M[1-(N*d-i+1)/(N-i+1)^{\frac{K}{2}}]$$

where $d = 1 - 1/M$.

As with the Cardenas expression, the function P' suffers its worst error when both M and K aproach N in value. Figure32 indicates, however, that even in this "worst case" , the

function P' is a very good approximation to P.

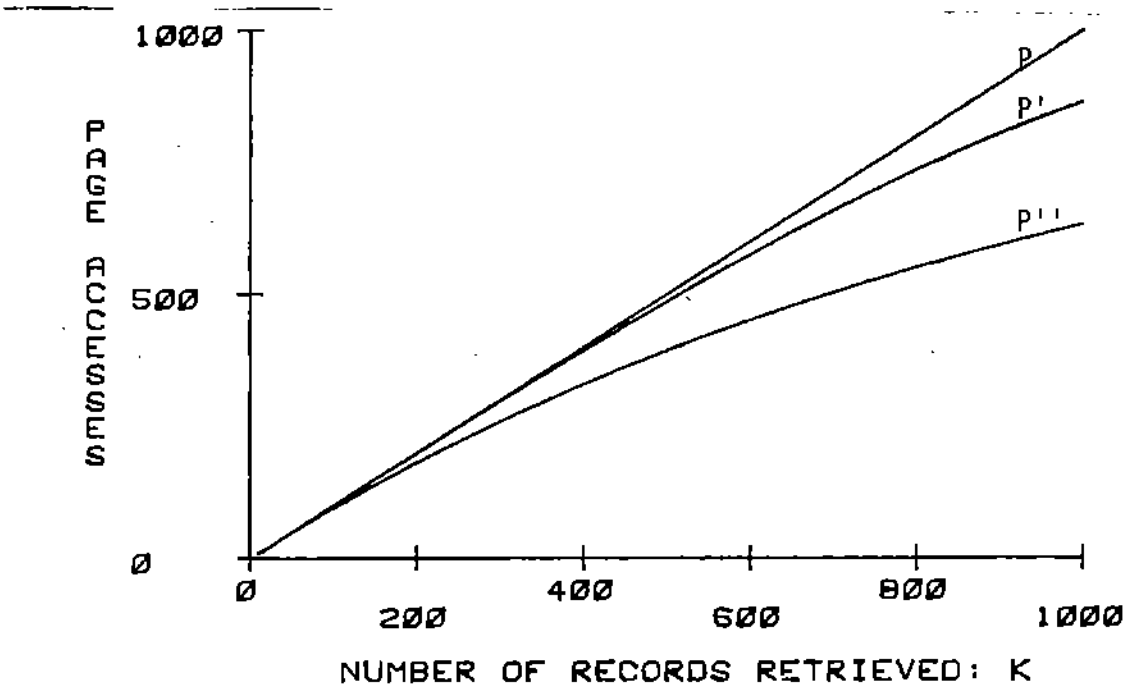


Figure 32. A comparison of the functions P, P', and P''.